



Les contrôleurs

Sources :

Partie 1

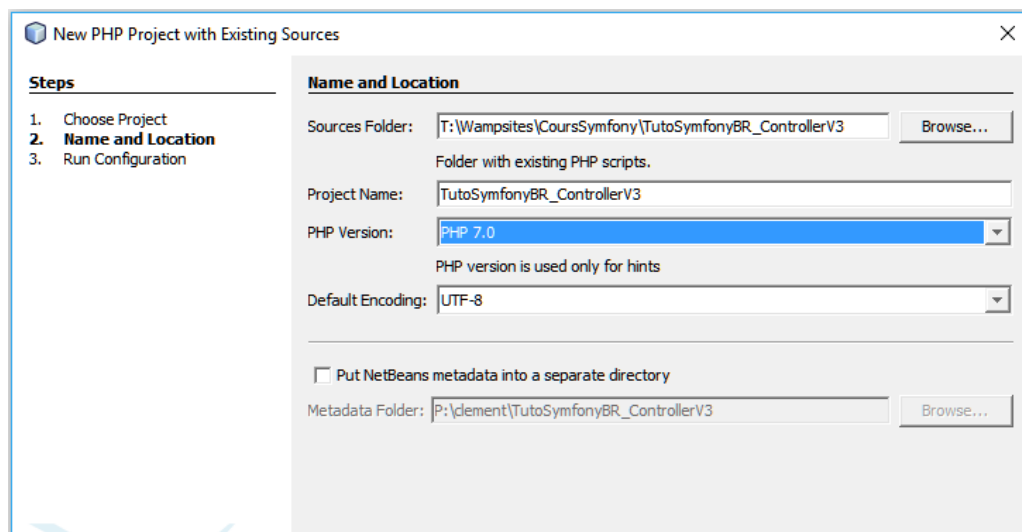
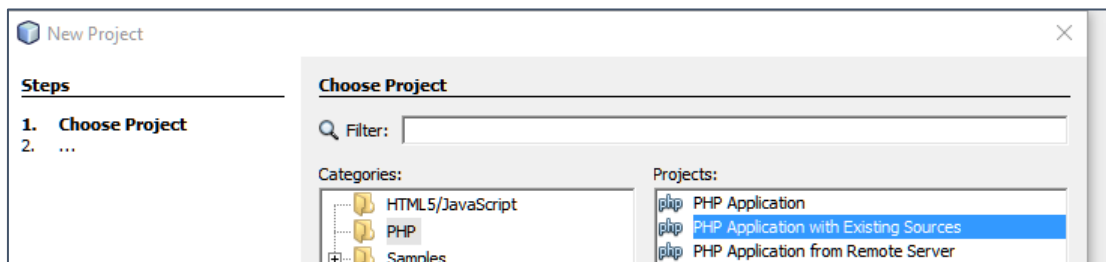
1. Préparation du travail

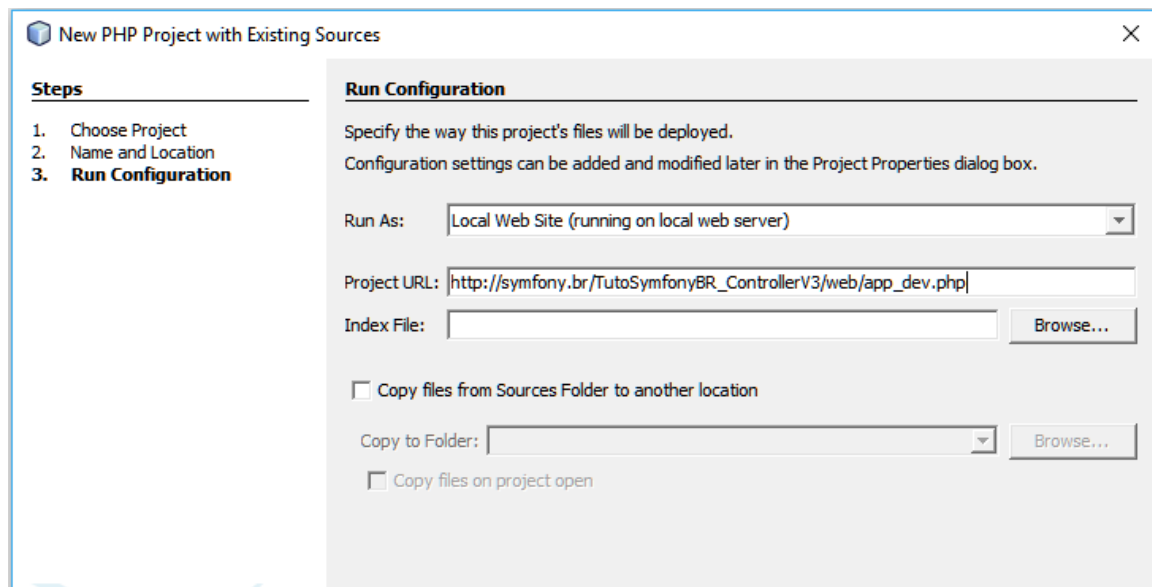
- ✓ Créer un nouveau projet
 - ✓ Nom : TutoSymfonyBR_ControllerV3

```
C:\WINDOWS\system32\cmd.exe
```

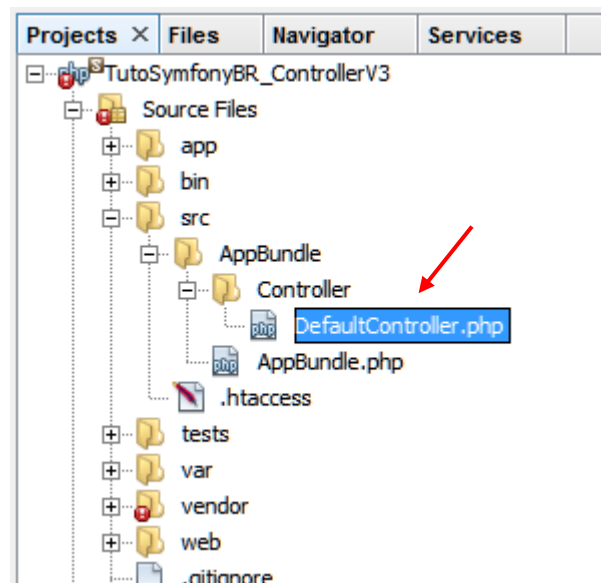
```
T:\Wampsites\CoursSymfony>php symfony new TutoSymfonyBR_ControllerV3
```

- ✓ Créer un nouveau projet NetBeans with existing sources ***TutoSymfonyBR_Ctrl***
Ne pas oublier !!



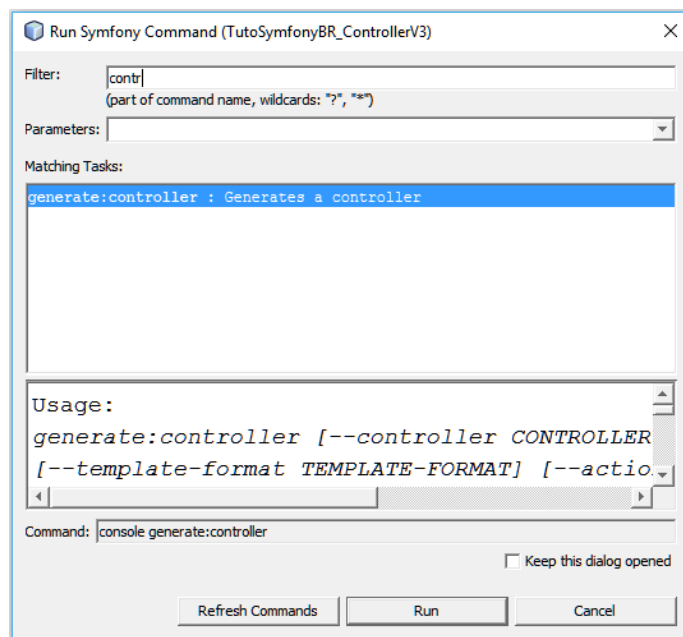
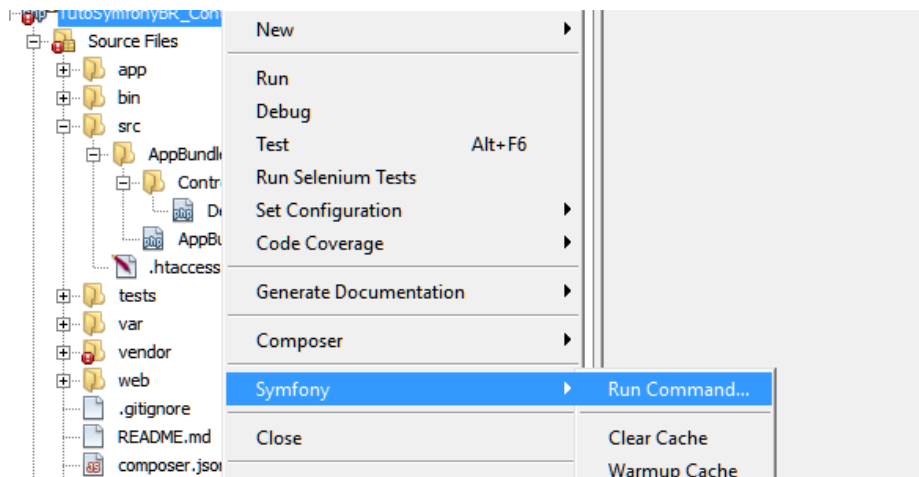


- ✓ On utilisera le bundle AppBundle
- ✓ Nettoyage :
 - Supprimer le contrôleur Controller/DefaultController.php



- ✓ Créer le contrôleur **TutorielCtrl** avec la commande **symfony**:

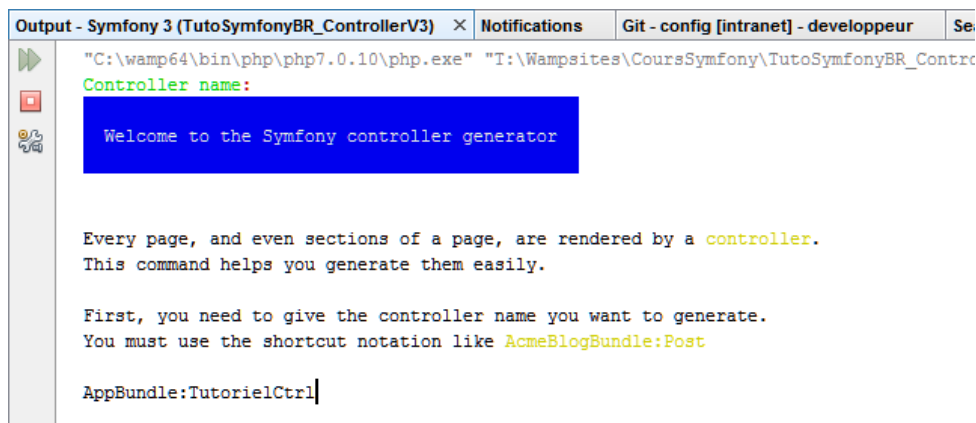
Ne rien y mettre dedans pour l'instant



Le premier paramètre à fournir est le nom du contrôleur

✓ Préfixé du nom du bundle

Donc AppBundle:TutorielCtrl



Deuxième information : le format des routes. On choisira le mode par défaut : annotation.

```
AppBundle:TutorielCtrl
Routing format (php, xml, yml, annotation) [annotation]:
Determine the format to use for the routing.
```

,

Puis le format du moteur de templates : twig. On remarque que l'on peut faire ses vues en PHP, mais ce n'est pas conseillé. On validera pour choisir le mode par défaut.

```
Template format (twig, php) [twig]: Determine the format to use for templating.
```

,

Enfin, on nous demande si on veut créer une action.

On va créer une action appelée hello (cela nous facilitera la tâche pour ce qui suit).

```
helloAction
Action route [/hello]: |
```

On confirme la route pour l'instant...

Il nous propose la vue hello.html.twig, on valide aussi ... encore une fois ça va nous faciliter la vie pour la suite....

```
- - - - -
New action name (press <return> to stop adding actions):
```

|

On valide ensuite, on ne veut pas créer de nouvelles actions. On rajoutera les suivantes par l'ide.

Enfin, il nous demande de valider la création du contrôleur.

```
Do you confirm generation [yes]?
```

Summary before generation

```
You are going to generate a "AppBundle:TutorielCtrl" controller
using the "annotation" format for the routing and the "twig" format
for templating
```

On valide ...

Notre contrôleur est créé :

Controller generation

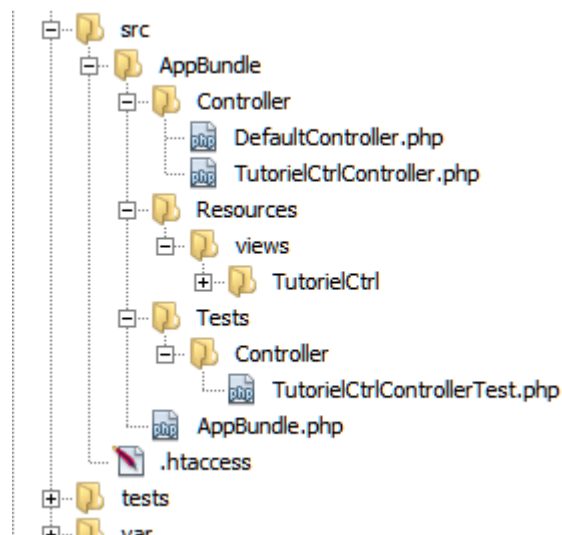
```
created .\src\AppBundle/Resources/views/TutorielCtrl/  
created .\src\AppBundle/Resources/views/TutorielCtrl/hello.html.twig  
created .\src\AppBundle\Controller\TutorielCtrlController.php  
created .\src\AppBundle\Tests\Controller/  
created .\src\AppBundle\Tests\Controller\TutorielCtrlControllerTest.php  
Generating the bundle code: OK
```

Everything is OK! Now get to work :).

Done.

|

Notre contrôleur a bien été créé ... mais pas QUE !!! :
Remarquez les dossiers qui ont été créés...



2. Exercice1 :

Modifier la route créée lors de la génération du nouveau contrôleur et qui va permettre d'intercepter l'URL :

http://symfony.br/TutoSymfonyBR_ControllerV3/web/app_dev.php/hello/Everybody

Everybody est une variable.

Cette URL permet d'afficher la page suivante :

... en Aucun cas vous devez appeler la vue hello.html.twig.

L'idée est de créer un objet Response acceptant en paramètre le bout de code HTML à afficher...

Bonjour Everybody

Vous risquez de vous heurter au message suivant ...



Indice :

Ne pas oublier le use en début de fichier pour pouvoir utiliser la classe Response.

Solution

```
namespace AppBundle\Controller;

use Symfony\Bundle\FrameworkBundle\Controller\Controller;
use Sensio\Bundle\FrameworkExtraBundle\Configuration\Route;
use Symfony\Component\HttpFoundation\Response;

class TutorielCtrlController extends Controller
{
    /**
     * @Route(
     *     path= "/hello/{nom}",
     *     name="TutorielCtrl_hello",
     *     requirements={"nom" : "[A-Za-z]+"})
     */
    public function helloAction($nom) {
        $page = "<html><body><H1> Bonjour " . $nom . "</H1></body></html>";
        return new Response($page);
    }
}
```

3. Exercice 2 : Redirection

Votre tâche va constituer à faire une redirection.

Pour cela, vous allez créer une nouvelle route qui va intercepter l'URL :

http://symfony.br/TutoSymfonyBR_ControllerV3/web/app_dev.php/ailleurs/Benoît

Benoît est une variable, pas de pattern imposé

La route va appeler l'action **ailleurs** du contrôleur TutorielCtrl.

Cette action va renvoyer sur la route **TutorielCtrl_hello** de l'exercice précédent.

Avant la redirection, vous concatèneriez au paramètre reçu une indication comme quoi vous avez bien redirigé l'URL :

`$param = $nom . " (après Redirect) !";`

\$nom étant le paramètre reçu par le contrôleur.

Indice :

Utiliser la méthode `redirectToRoute` de la classe Controller.

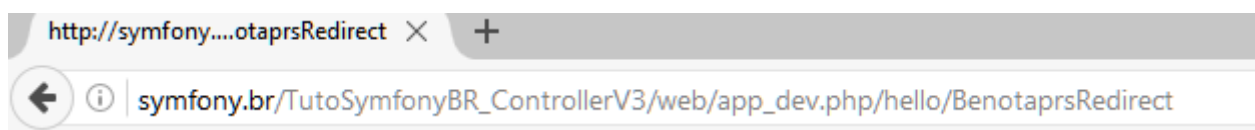
En faisant ceci, vous risquez d'avoir le message :



Normal, le paramètre n'est pas conforme au pattern du paramètre réclamé par la route TutorielCtrl_hello

Vous placerez donc, avant la rediriger, une expression régulière qui supprime tous les caractères non autorisés.

Au final, vous devriez obtenir :



Bonjour BenotaprsRedirect

Indice :

Utiliser la méthode redirectToRoute de la classe Controller.



Remarquez l'URL, elle a changé. Le paramètre est modifié et correspond au pattern de la route.

Solution

Contrôleur TutorielCtrl :

```
/**
 * @Route(
 *     path= "/ailleurs/{nom}",
 *     name="TutorielCtrl_ailleurs"
 * )
 */
public function ailleursAction($nom) {
    // On modifie le paramètre
    // ceci pourrait se faire après une action sur la BD par exemple.
    $param = $nom . " (après Redirect) !";
    //Avant d'envoyer, on met le paramètre en conformité avec le patter exigé
    // par la route appelée
    $param = preg_replace("#[^a-zA-Z]#", "", $param);
    return $this->redirectToRoute('TutorielCtrl_hello', array('nom' => $param));
}
```

4. Exercice 3 : transfert

Votre tâche va constituer à faire une redirection.

Pour cela, vous allez créer une nouvelle route qui va intercepter l'URL :

http://localhost/.../app_dev.php/ctrl/transferer/Benoit

benoit est une variable, pas de pattern imposé

La route va appeler l'action **transfert** du contrôleur TutorielCtrl.

Cette action va appeler l'action Hello du contrôleur TutorielCtrl

Vous concatènerez au paramètre reçu une indication comme quoi vous avez bien forwardé l'URL

Vous pourrez par exemple obtenir le résultat suivant après avoir récupéré la date et l'heure du serveur web :



Indice :

Utiliser la méthode forward de la classe Controller.



Vous remarquerez que l'URL n'a pas changé, mais que c'est bien l'action hello du contrôleur qui a été appelée. On n'est pas repassé par la route. Si tel avait été le cas, elle aurait déclenché une erreur à cause du pattern qui n'était pas respecté.

Solution

Contrôleur TutorielCtrl :

```
/**
 * @Route (
 *     path= "/transferer/{nom}",
 *     name="TutorielCtrl_transferer",
 *     requirements={"nom" : "[A-Za-z]+"}
 * )
 */
public function transfererAction($nom) {
    // On modifie le paramètre
    // ceci pourrait se faire après une action sur la BD par exemple.
    $param = $nom . " (Transféré à " . date("d/m/Y - H:i") . " )";
    return $this->forward('AppBundle:TutorielCtrl:hello', array('nom' => $param));
}
```

Partie 2

1. Présentation du travail

On va faire un petit exercice qui va nous permettre d'introduire les formulaires et twig.

2. Préparation du travail

On vous demande de créer la route TutorielCtrl_testfrm dans le bundle AppBundle du projet TutoSymfonyBR.

Cette route doit intercepter l'url app/dev/testfrm/{nom}, nom étant une variable n'acceptant que des lettres dont la première est une majuscule et les autres des minuscules.

Cette route appelle l'action testfrm du contrôleur TutorielCtrlController.

La méthode correspondant à cette action accepte en paramètre, en plus du nom, l'objet Request.

Fichiersrc\Bdls\CtrlBundle\Controller\TutorielCtrlController.php :

```
/**
 * @Route (
 *     path= "/testfrm/{nom}",
 *     name="TutorielCtrl_testfrm",
 *     requirements={"nom" : "[A-Za-z]+"}
 * )
 */
public function testfrmAction($nom, Request $request) {
}
```

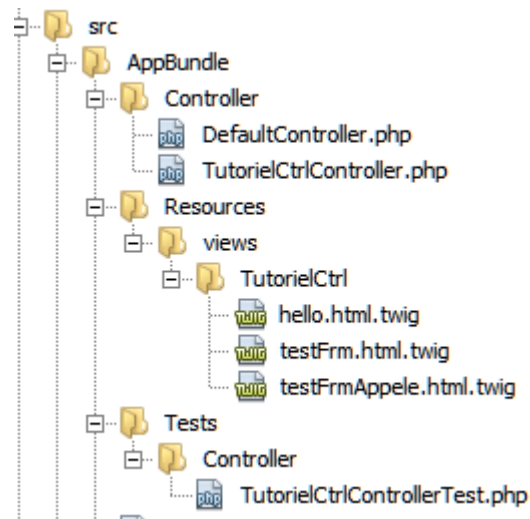
Notez les paramètres ...

- ✓ \$nom : la variable nom passée dans l'url
- ✓ L'objet \$request de la classe Request qui va nous permettre de récupérer pas mal d'informations notamment celles renvoyées par le formulaire que l'on va créer

Le formulaire :

Dans le dossier ...src\Bdls\CtrlBundle\Resources\views\TutorielCtrl :

- ✓ Créer les dossiers Resources/views/TutorielCtrl
- ✓ Créer la vue twig appelée testfrm.html.twig



Le contenu de la vue ::

```
{# empty Twig template #}
<html>
  <body>
    {{ form(leFormulaire) }}
  </body>
</html>
```



Rien de bien exceptionnel, mais on fera mieux !!!



La fonction form de twig n'existe que depuis symfony 2.3

Saisir le code suivant dans la méthode testfrmAction du controller :

```
public function testfrmAction($nom, Request $request) {
    $formulaire = $this->createFormBuilder()
        ->add('znom', TextType::class)
        ->add('zsprenom', TextType::class, array('label' => 'Saisir votre prénom :'))
        ->add('zsage', IntegerType::class, array('label' => 'saisir votre âge : ',
            'invalid_message' => 'Veuillez saisir un entier'))
        ->add('sexe', ChoiceType::class, array('choices' => array('Femme' => 'F', 'Homme' => 'H'), 'expanded' => true, 'multiple' => false))
        ->add('sauvegarder', SubmitType::class)
        ->getForm();

    return $this->render('AppBundle:TutorielCtrl:testFrm.html.twig', array('leFormulaire' => $formulaire->createView(), 'nom' => $nom));
}
```

Indice :

Ne pas oublier de mettre le use qui vont bien

Tester l'Url suivante :

http://symfony.br/TutoSymfonyBR_ControllerV3/web/app_dev.php/testfrm/Dupont

Vous devriez voir ça :

AppBundle:TutorielCtrl:he... X +

← | symfony.br/TutoSymfonyBR_ControllerV3/web/app_dev.php/testfrm/Dupont

Premier formulaire

Zsnom

Saisir votre prénom :

saisir votre âge :

Sexe

☐ Femme ☐ Homme

... c'est un bon début ...

Vous remarquerez plusieurs choses :

```
$formulaire = $this->createFormBuilder()
```

- ✓ La classe Controller possède la méthode createFormBuilder qui permet de créer un objet de la classe FormBuilder
- ✓ Cette classe va permettre de définir les différents champs qui vont composer le formulaire
- ✓ La classe FormBuilder implémente la méthode add() qui permet de décrire les différents champs du formulaire.

```
->add('zsnom', TextType::class)
->add('zsprenom', TextType::class, array('label' => 'Saisir votre prénom :'))
->add('zsage', IntegerType::class, array('label' => 'saisir votre âge : ',
    'invalid_message' => 'Veuillez saisir un entier'))
->add('sexe', ChoiceType::class, array('choices' => array('Femme' => 'F', 'Homme' => 'H')))
->add('sauvegarder', SubmitType::class)
```

- ✓ La classe FormBuilder implémente la méthode getForm() () qui permet de créer le formulaire

```
->getForm();
```

- ✓ Enfin, pour afficher le formulaire, on va utiliser la méthode render de la classe Controller qui va générer l'objet Response à renvoyer, cet objet Response qui va permettre d'afficher la page

```
return $this->render('AppBundle:TutorielCtrl:testFrm.html.twig',
    array('leFormulaire' => $formulaire->createView(), 'nom' => $nom));
```

On remarquera :

Que l'on appelle le template testfrm.html.twig situé dans le dossier Tutoriel du dossier Views du bundle.

Qu'en deuxième paramètre de la méthode render, on va lui passer un tableau associatif dans lequel on va passer à l'indice 'leFormulaire', l'objet FormBuilder que l'on vient de créer, après lui avoir demandé d'afficher le formulaire

- ✓ En effet, la classe FormBuilder implémente la méthode createView() qui va faire appel à un certain nombre de helpers pour afficher le formulaire

```
return $this->render('AppBundle:TutorielCtrl:testfrm.html.twig',
    array('leFormulaire' => $formulaire->createView()));
```

Le template appelé est testfrm.html.twig :

```
{# empty Twig template #}
<html>
    <body>
        {{ form(leFormulaire) }}
    </body>
</html>
```

On voit que l'on est passé par la fonction form de twig pour afficher l'objet FormBuilder passé en paramètre au template.

On n'est pas obligés de passer uniquement des objets FormBuilder tout préparés. On peut aussi passer au formulaire des variables Par exemple, on n'a pas utilisé le paramètre \$nom passé par l'URL.

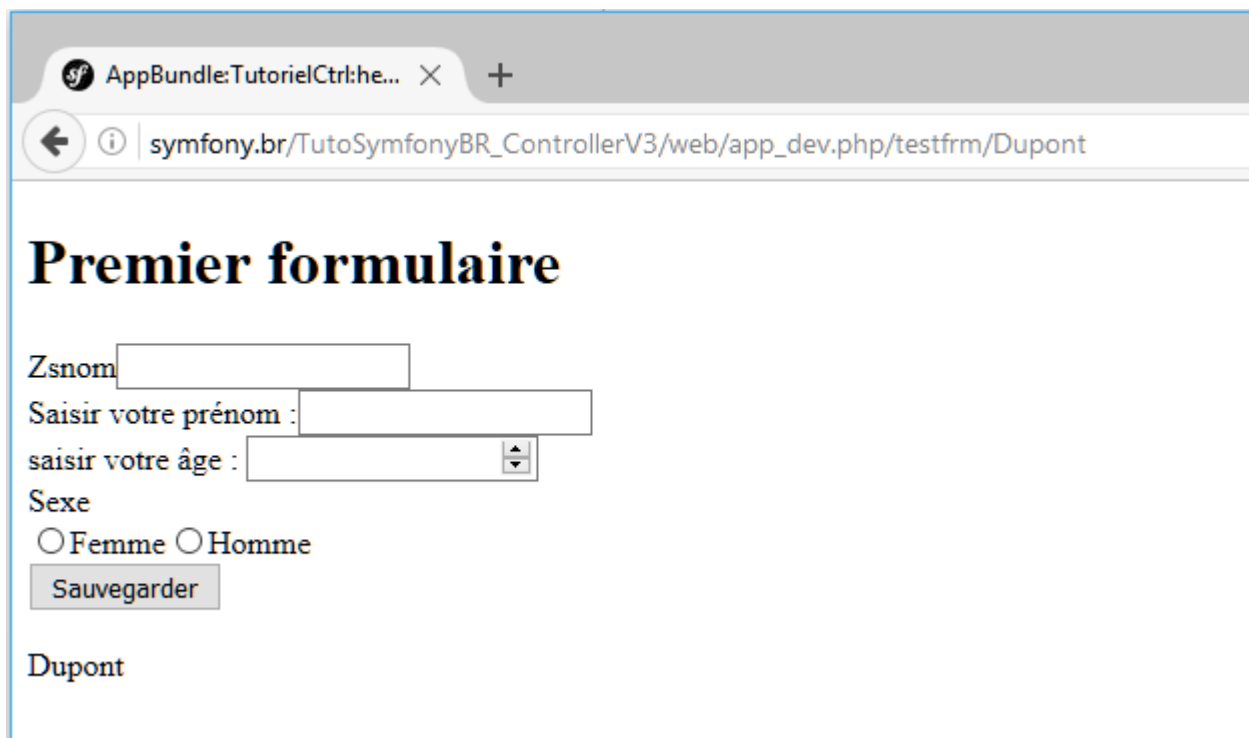
Modifier le tableau passé en paramètre de la méthode render de l'objet Controller :

```
->getForm();
return $this->render('BdlsCtrlBundle:TutorielCtrl:testfrm.html.twig', array('leFormulaire' => $formulaire->createView(), 'nom'=>$nom));
```

Et modifier le template pour accueillir cette variable nom :

```
{# empty Twig template #}
<html>
    <body>
        {{ form(leFormulaire) }}
        <p>{{ nom }}</p>
    </body>
</html>
```

Et voici l'affichage :



The screenshot shows a web browser window with the address bar displaying 'symfony.br/TutoSymfonyBR_ControllerV3/web/app_dev.php/testfrm/Dupont'. The page title is 'Premier formulaire'. The form contains the following elements:

- A label 'Zsnom' followed by a text input field.
- A label 'Saisir votre prénom :' followed by a text input field.
- A label 'saisir votre âge :' followed by a text input field with a spinner.
- A label 'Sexe' followed by two radio buttons: 'Femme' and 'Homme'.
- A 'Sauvegarder' button.
- The name 'Dupont' displayed below the form.



Amusez vous à tester...

Avez-vous vu le message ? Bluffant non ?

Pas mal non pour un début !!!



Oupps ... On n'a pas vu le code HTML généré Allez vérifier !!

```

<!DOCTYPE html>
<html>
  <head></head>
  <body>
    <h1>Premier formulaire</h1>
    <form name="form" method="post">
      <div id="form">
        <div>
          <label class="required" for="form_zsnom">Zsnom</label>
          <input id="form_zsnom" name="form[zsnom]" required="required" type="text">
        </div>
      </div>
      <div>
        <label class="required" for="form_zsage">saisir votre âge :</label>
        <input id="form_zsage" name="form[zsage]" required="required" type="number">
      </div>
      <div>
        <label class="required">Sexe</label>
        <div id="form_sexe">
          <input id="form_sexe_0" name="form[sexe]" required="required" value="F" type="radio">
          <label class="required" for="form_sexe_0">Femme</label>
          <input id="form_sexe_1" name="form[sexe]" required="required" value="H" type="radio">
          <label class="required" for="form_sexe_1">Homme</label>
        </div>
      </div>
    </form>
  </body>
</html>

```

... et plein d'autres choses....



Observons toutefois la balise form :

```

<html>
  <body>
    <form name="form" method="post">
      <div id="form">
        <div>

```

Et les noms des champs :

```

<div>
  <label class="required" for="form_zsage">saisir votre âge :</label>
  <input id="form_zsage" name="form[zsage]" required="required" type="number">
</div>

```

Que remarquez-vous ?

- ✓ On remarque aussi que l'attribut action du formulaire n'est pas précisé Cela veut tout simplement dire que le formulaire, quand il sera validé, exécutera à nouveau la méthode du contrôleur qui l'a invoqué !!!

- ✓ Il est aussi demandé d'utiliser la méthode post pour soumettre le formulaire



Il est temps maintenant de récupérer nos données

Modifiez le code de la méthode frmtestAction et rajoutez ceci :

```
->getForm();

//On va établir le lien entre le formulaire et le Request
// l'objet $formulaire contient notamment toutes les valeurs passées en get, post,...
$formulaire->handleRequest($request);

// On va tester si le formulaire est valide, c'est à dire s'il respecte
// les règles de validations qu'on verra plus loin.
if ($formulaire->isValid()) {

    //On récupère les données POST dans le tableau $tab
    $tab=$formulaire->getData();
    // On envoie les données sur le nouveau formulaire frmAppele
    return $this->render('AppBundle:TutorielCtrl:testfrmappele.html.twig',
        array('donneesformulaire' => $tab));
}

// on va rediriger vers un nouveau formulaire via une nouvelle route !!!
```

On remarque la méthode handleRequest(\$request) du de l'objet FormBuilder qui

- ✓ Ne fait rien lors du chargement initial du formulaire
- ✓ Récupère les données soumises quand le formulaire revient après l'action sur le bouton submit

Ensuite, il est important de vérifier que les données saisies sont valides....

Si les données sont valides, on récupère dans \$tab les données postées (voir le nom des champs du formulaire....)

```
<div>
<label class="required" for="form_zsage">saisir votre âge </label>
<input id="form_zsage" name="form[zsage]" required="required" type="number">
```

form est donc LE tableau contenant les données saisies du formulaire.

Là, on fait ce que l'on veut.... J'ai choisi
De récupérer les données du formulaire :



```
//On récupère les données POST dans le tableau $tab
$tab=$formulaire->getData();
```

d'appeler un nouveau formulaire pour afficher les données (testfrmappele.html.twig). On lui passe en paramètre le tableau des données saisies.

```
// On envoie les données sur le nouveau formulaire frmAppele
return $this->render('AppBundle:TutorielCtrl:testfrmappele.html.twig',
    array('donneesformulaire' => $tab));
}
```

Il n'y a plus qu'à tester !!!

URL : http://localhost/.../web/app_dev.php/ctrl/testfrm/Dupont



Premier formulaire

Zsnom

Saisir votre prénom :

saisir votre âge :

Sexe
☐ Femme ☒ Homme

Dupont



Les affichages peuvent changer en fonction du navigateur

Résultat après soumission :

Bond

James

99

H

_rFLr7wbRv1gtXpG6a4N3-s1w0sAtmG0o2dm5qWfeJM

Oups,....

Pour bien comprendre le fonctionnement, je vous propose de faire une exécution pas à pas et de voir par où passe le programme :

Arrêtons-nous un instant, en mode xdebug, examinez les variables :

\$formulaire

```
Symfony\Component\Form\Form object {
  config => Symfony\Component\Form\FormBuilder object {
    *Symfony\Component\Form\FormConfigBuilder*nativeRequestProcessor => null
    *Symfony\Component\Form\FormConfigBuilder*allowedMethods => array(5) (
      [0] => (string) GET
      [1] => (string) PUT
      [2] => (string) POST
      [3] => (string) DELETE
      [4] => (string) PATCH
    )
    children => array(0)
    unresolvedChildren => array(0)
    locked => (bool) true
    *Symfony\Component\Form\FormConfigBuilder*dispatcher => Symfony\Component\EventDispatcher\ImmutableEventDispatcher object {
      dispatcher => Symfony\Component\EventDispatcher\EventDispatcher object {
      }
    }
    *Symfony\Component\Form\FormConfigBuilder*name => (string) form
    *Symfony\Component\Form\FormConfigBuilder*propertyPath => null
    *Symfony\Component\Form\FormConfigBuilder*mapped => (bool) true
    *Symfony\Component\Form\FormConfigBuilder*byReference => (bool) true
    *Symfony\Component\Form\FormConfigBuilder*inheritData => (bool) false
    *Symfony\Component\Form\FormConfigBuilder*compound => (bool) true
    *Symfony\Component\Form\FormConfigBuilder*type => Symfony\Component\Form\Extension\DataCollector\Proxy\ResolvedTypeDataCollectorProxy object...
```

\$tab

```
array(4) (
  [zsnom] => (string) Bond
  [zsprenom] => (string) James
  [zsage] => (int) 99
  [sexe] => (string) H
)

$tab=$formulaire->ge
```



Les contrôles côté client c'est bien Mais ça ne dispense pas de les refaire côté serveur !!!