

Symfony 5 : Les formulaires

Benoît Roche
@rocheb83

LES FORMULAIRES

LES FORMULAIRES

Pour illustrer le cours, nous travaillerons avec

- L'entity **Demande** qui représente des demandes de renseignements.
- Le controlleur **DemandeController**
- Le template article/**editdemande.html.twig**






L'idée est de présenter ici comment faire en sorte de construire des formulaires permettant de créer et d'éditer(voir/modifier) des articles.

Plusieurs pistes de développement seront présentées dans ce diaporama.

LES FORMULAIRES

Pour illustrer le cours, nous travaillerons avec

- L'entity **Demande** qui représente des demande de renseignements de personnes
- Cette entity a les caractéristiques suivantes

+ Demande		
	- id	int
	- titre	string
	- question	text
	- mail	string
	- dateCreation	DateTime

```
class Demande
{
    #[ORM\Id]
    #[ORM\GeneratedValue]
    #[ORM\Column]
    private ?int $id = null;

    #[ORM\Column(length: 100)]
    private ?string $titre = null;

    #[ORM\Column(type: Types::TEXT)]
    private ?string $question = null;

    #[ORM\Column(length: 120)]
    private ?string $mail = null;

    #[ORM\Column(type: Types::DATETIME_MUTABLE)]
    private ?\DateTimeInterface $datecreation = null;
```

LES FORMULAIRES

Le formulaire est le principal composant permettant d'interagir avec un utilisateur.

On va construire des formulaires pour :

- ✓ Afficher des données
- ✓ Permettre la saisie des données
- ✓ Valider les données soumises au serveur d'application,
- ✓ Mapper les données du formulaires en objets
- ✓ Persister ces objets dans une base de données



```
aire50> symfony composer require symfony/form
```

Indispensable

LES FORMULAIRES

Dans tous les cas, nous aurons à :

- ✓ Construire le formulaire
- ✓ Mettre le formulaire à disposition de l'application grâce à un moteur de templates (twig en général) afin que l'utilisateur dernier puisse interagir avec les données et soumettre le formulaire
- ✓ Traiter le formulaire soumis pour valider les données et les traiter. Par exemple les persister en BD

On pourra construire les formulaires de deux façons :

- ✓ Par la méthode **createFormBuilder** de la classe AbstractController
- ✓ Par l'intermédiaires de **classes de formulaires**



CRÉATION DE FORMULAIRE DANS LE CONTRÔLEUR

LES FORMULAIRES

Formulaires dans le contrôleur:

L'idée est :

- ✓ de construire le formulaire dans le contrôleur
- ✓ et de l'envoyer à twig qui l'affichera

On utilisera la méthode

`AbstractController::createFormBuilder`

L'idée est de créer un formulaire qui sera lié (bind) à un objet d'une classe Entity



C'est ce que vous avez fait au TP précédent !!!

LES FORMULAIRES

Formulaires dans le contrôleur:

```
#[Route('/demande', name: 'demande')]
public function demandeContact(Request $request)
{
    $contact = new Contact();

    $form = $this->createFormBuilder($contact)
        ->add('titre')
        ->add('nom')
        ->add('mail')
        ->add('tel')
        ->getForm();

    return $this->render('contact/contact.html');
```

```
{% extends 'base.html.twig' %}

{% block title %} {{ path('article_new') }} {% endblock %}

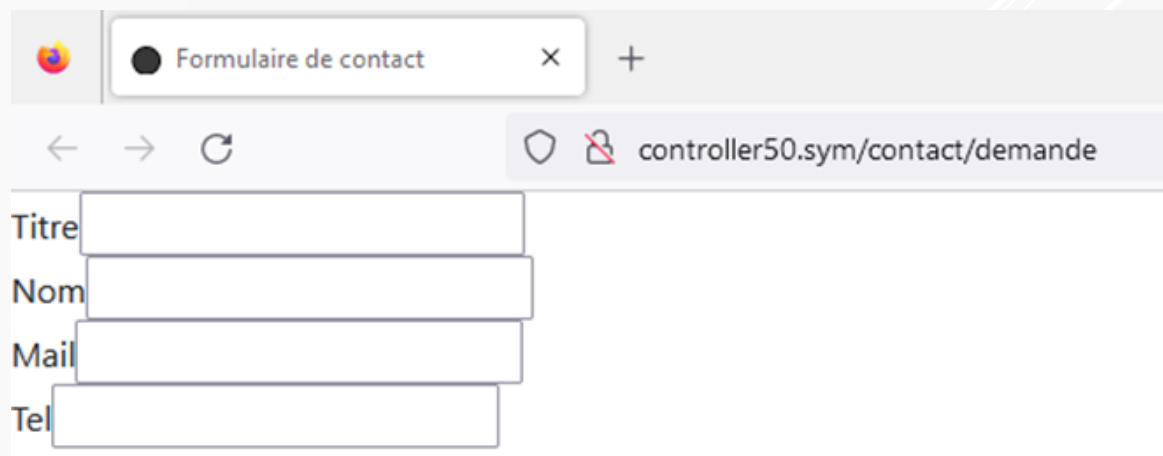
{% block body %}

    <h1> Nouvel article </h1>

    {{ form(formArticle) }}

{% endblock %}
```

Très simple !!!



The screenshot shows a web browser window with the title "Formulaire de contact". The address bar displays "controller50.sym/contact/demande". The form contains four input fields labeled "Titre", "Nom", "Mail", and "Tel".

<http://controller50.sym/contact/demande>

LES FORMULAIRES

Formulaires dans le contrôleur:

```
public function demandeContact(Request $request, GestionCont  
    $contact = new Contact();  
    $form = $this->createFormBuilder($contact)  
        ->add('titre')  
        ->add('nom')  
        ->add('mail')  
        ->add('tel')  
        ->getForm();  
    return $this->render('contact/contact.html.twig',  
        ['formContact' => $form->createView(),  
        'titre' => 'Formulaire de contact',  
    ]);
```

On a indiqué les attributs de la classe à afficher dans le formulaire

getForm() renvoie un objet représentant le formulaire configuré

createView() va créer un objet FormView exploitable par twig !

<http://controller50.sym/contact/demande>

LES FORMULAIRES

Formulaires dans le contrôleur:

```
{% extends 'base.html.twig' %}

{% block title %} {{ path('article_new') }} {% endblock %}

{% block body %}

    <h1> Nouvel article </h1>

    {{ form(formArticle) }}

{% endblock %}
```

<http://premierprojet40.sym/article/new>

La fonction twig **form()** permet de restituer un formulaire entier Avec éventuellement les messages d'erreur.



Le résultat n'est pas très ergonomique ni très beau mais symfony a été capable :

- ✓ De créer un champ appropriés au attributs de la classe
- ✓ De créer un formulaire .

Les champs du formulaire seront mappés sur les attributs de l'entité Contact

LES FORMULAIRES

Formulaires dans le contrôleur:

On peut faire mieux en personnalisant un peu plus de paramètres c'est-à-dire :

- ✓ Forcer le type de formulaire voulu (contrôle, sous formulaire...)
- ✓ Et fournir des paramètres HTML



C'est ce que vous avez fait au TP précédent !!!

LES FORMULAIRES

Formulaires dans le contrôleur:

Exemples :

```
->add('titre', ChoiceType::class, array(  
    'choices' => array(  
        'Monsieur' => 'M',  
        'Madame' => 'F',  
    ), 'multiple' => false,  
    'expanded' => true,  
))
```

Titre
☐ Monsieur ☐ Madame

```
->add('nom', TextType::class,  
    array(  
        'label' => 'Nom : ',  
        'required' => true,  
        'attr' => ['placeholder' => 'votre nom'],  
    ),
```

Nom :

```
->add('mail', EmailType::class,  
    array(  
        'label' => 'Mail : ',  
        'required' => true,  
    ),  
))
```

Mail :



Symfony vous a aidé
dans l'écriture de code !

CRÉATION D'UNE CLASSE DE FORMULAIRE

LES FORMULAIRES

Types de formulaires :

Dans Symfony, il faut bien comprendre la notion de **types de formulaires** :

Il n'y a pas de différence entre formulaire et champ de formulaire comme en HTML.

Dans Symfony, **TOUT** est Type de Formulaire :

- ✓ **un champ de formulaire unique** : TextType équivalent d'un élément `<input type="text">` HTML
- ✓ **un groupe de champs**. Par exemple le groupe de champs permettant la saisie d'une adresse postale : AdressePostaleType)
- ✓ **L'ensemble** des champs d'une classe de type Entity (exemple : EmployeType).



Un type de formulaire = Une classe xxxType

LES FORMULAIRES

Formulaires dans une classe Formulaire :



Solution à privilégier

On crée un type de formulaire avec la commande symfony :

make:form

Cette commande va créer un objet de la classe XxxType qui va hériter de la classe AbstractType

Cette classe XxxType contiendra la méthode **buildForm** qui fera la même chose que la méthode **CreateFormBuilder** de la classe AbstractController vue précédemment.



*Par convention ces classes sont suffixées par le mot **Type** (ex ArticleType)*

LES FORMULAIRES

Formulaires dans une classe Formulaire :

```
PS T:\Wampsites\CoursSymfony\formulaire50> symfony console make:form
```

```
The name of the form class (e.g. GentlePizzaType):
```

```
> Demande
```

```
The name of Entity or fully qualified model class name that the new form will be bound to (empty for none):
```

```
> Demande
```

```
created: src/Form/DemandeType.php
```

Success!

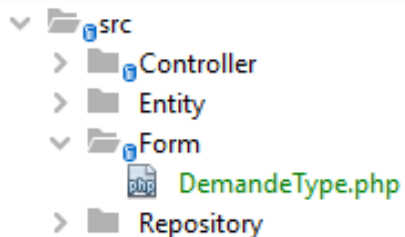
Next: Add fields to your form and start using it.

Find the documentation at <https://symfony.com/doc/current/forms.html>

```
PS T:\Wampsites\CoursSymfony\formulaire50>
```

Vous fournirez

- le nom de la classe Form
- Le nom de l'entity qui sera mappée, *mais ce n'est pas obligatoire*



Le dossier Form a été créé
Il héberge le formulaire Demande Type

LES FORMULAIRES

Formulaires dans une classe Formulaire :

```
class DemandeType extends AbstractType
{
    public function buildForm(FormBuilderInterface $builder, array $options): void
    {
        $builder
            ->add('titre')
            ->add('question')
            ->add('mail')
            ->add('datecreation')
        ;
    }

    public function configureOptions(OptionsResolver $resolver): void
    {
        $resolver->setDefaults([
            'data_class' => Demande::class,
        ]);
    }
}
```

Et voilà un formulaire réutilisable dans plusieurs contrôleurs

On n'est pas non plus obligés de mettre tous les champs
Voir plus loin



On va l'améliorer

LES FORMULAIRES

Formulaires dans une classe Formulaire :

Utilisation dans le contrôleur : on utilise la méthode `createForm` de la classe `AbstractController` :

```
#[Route('/demande', name: 'demande_')]
class DemandeController extends AbstractController
{
    #[Route('/creer', name: 'creer')]
    public function creer() : Response {
        $demande = new Demande();
        // On crée le formulaire
        $form= $this->createForm(DemandeType::class, $demande);
        // on envoie la réponse au navigateur sous forme de formulaire
        return $this->render('demande/creer.html.twig',
            ['formDemande'=>$form->createView()]);
    }
}
```

En paramètre on donne à la méthode :

- ✓ La classe `XxxType`
- ✓ L'entity mappée

En paramètre on passe le formulaire créé

LES FORMULAIRES

Formulaires dans une classe Formulaire :

On n'a plus qu'à créer la vue twig et à tester !!!

```
{% extends 'base.html.twig' %}

{% block title %}Demande{% endblock %}

{% block body %}
    <h1>Nouvelle demande</h1>
    {{ form(formDemande) }}
{% endblock %}
```

<http://formulaire50.sym/demande/creer>



The screenshot shows a web browser window with the address bar displaying 'formulaire50.sym/demande/creer'. The page title is 'Nouvelle demande'. The form contains the following fields:

- Titre**: A text input field.
- Question**: A text area field.
- Mail**: A text input field.
- Datecreation**: A date and time selection field with dropdown menus for month (Jan), day (1), year (2017), hour (00), and minute (00).

symfony a été capable de créer :

- ✓ Un champ textarea pour la question
- ✓ Un champ date heure pour la date de création.

SOUS FORMULAIRES NON RATTACHÉS À UNE ENTITY

Formulaires sans entity

Il est possible de créer des formulaires xxxType sans y rattacher d'entity.

Ceci revient à créer un composant réutilisable.

Exemple :

+ Client	
<input type="text"/>	- id
<input type="text"/>	- nom
<input type="text"/>	- prenom
<input type="text"/>	- cp
<input type="text"/>	- ville

+ Entrepot	
<input type="text"/>	- id
<input type="text"/>	- nom
<input type="text"/>	- adresse
<input type="text"/>	- cp
<input type="text"/>	- ville



On peut créer un type CpVilleType que l'on pourra utiliser dans les formulaires de gestion des clients et des entrepôts

LES FORMULAIRES

Formulaires sans entity:

on peut créer une classe xxxType vierge dans laquelle on va pouvoir rajouter ses propres champs :

```
PS T:\Wampsites\CoursSymfony\formulaire50> symfony console make:form
```

```
The name of the form class (e.g. AgreeablePuppyType):  
> CpVille
```

```
The name of Entity or fully qualified model class name that the new form will be bound to (empty for none):  
>
```

```
Created: src/Form/CpVilleType.php
```

Success!

Next: Add fields to your form and start using it.

Find the documentation at <https://symfony.com/doc/current/forms.html>

```
PS T:\Wampsites\CoursSymfony\formulaire50> █
```

Construction du formulaire **CpVilleType**
... qui est vierge !

```
public function buildForm(FormBuilderInterface $builder  
{  
    $builder  
        ->add('field_name')  
    ;  
}
```

LES FORMULAIRES

Formulaires dans une classe Formulaire :

On n'a plus qu'à

- ✓ Ajouter les champs que l'on veut faire apparaître dans le formulaire

```
class CpVilleType extends AbstractType
{
    public function buildForm(FormBuilderInterface $builder, array $options): void
    {
        $builder
            ->add('cp')
            ->add('ville')
        ;
    }
}
```

C'est ce même type (CpVilleType) qui apparaîtra dans les formulaires

- ✓ ClientType
- ✓ EntrepotType

LES FORMULAIRES

Formulaires dans une classe Formulaire :

On n'a plus qu'à

- ✓ Ajouter les champs que l'on veut faire apparaître dans le formulaire

```
public function buildForm(FormBuilderInterface $builder  
    $builder  
        ->add('nom')  
        ->add('prenom')  
        ->add($builder->create('cpVille',  
                                FormType::class,  
                                ['mapped' => true])  
            ->add('cp')  
            ->add('ville')  
        )  
        ->add('Creer', SubmitType::class)
```

client

```
public function buildForm(FormBuilderInterface $builder  
    $builder  
        ->add('nom')  
        ->add('adresse')  
        ->add($builder->create('cpVille',  
                                FormType::class,  
                                ['mapped' => true])  
            ->add('cp')  
            ->add('ville')  
        )  
        ->add('Creer', SubmitType::class);  
    }
```

entrepôt

LES FORMULAIRES

Formulaires dans une classe Formulaire :

Pour que le mappage puisse se faire automatiquement, il suffit d'écrire les getter/setter du composant du sous formulaire dans les classes entity

```
->add($builder->create('cpVille',  
    FormType::class,  
    ['mapped' => true]))
```

```
public function getCpVille() :?string {  
    return null;  
}  
  
public function setCpVille(array $params) :void {  
    $this->cp=$params['cp'];  
    $this->ville=$params['ville'];  
}
```



Le getter : Pour permettre l'affichage du formulaire

Le setter : Pour permettre le mapping au retour du formulaire

LES FORMULAIRES

Formulaires dans une classe Formulaire :

formulaire50.sym/client/creer

Nouveau client

Nom Dupont
Prenom Jean
Cp ville
Cp 83000
Ville Toulon
Creer

Result Grid Filter Rows: Edit:

	id	nom	prenom	cp	ville
▶	1	Dupont	Jean	83000	Toulon
*	NULL	NULL	NULL	NULL	NULL

formulaire50.sym/entrepot/creer

Nouvel entrepôt

Nom Ets Blanchard
Adresse Avenue de Pau
Cp ville
Cp 33000
Ville Bordeaux
Creer

	id	nom	adresse	cp	ville
▶	1	Ets Blanchard	Avenue de Pau	33000	Bordeaux
*	NULL	NULL	NULL	NULL	NULL

<http://formulaire50.sym/client/creer> <http://formulaire50.sym/entrepot/creer>



On a toujours bien entendu la possibilité de personnaliser l'affichage... ouf

SOUS-FORMULAIRE GÉRANT LES RELATIONS ENTRE ENTITIES

LES FORMULAIRES

Formulaires sans entity

Il est possible de créer des formulaires xxxType en y rattachant une entity.

Exemple :



On va créer le formulaire permettant un article et sa famille d'appartenance



Quelle est la signification de ce modèle ?

LES FORMULAIRES

Formulaires : les sous formulaires

Représentation de l'association bidirectionnelle entre les Entities Article et Famille

Dans l'entity Article :

```
# [ORM\ManyToOne (inversedBy: 'articles')]
# [ORM\JoinColumn(nullable: false, name: 'idfamille')]
private ?Famille $famille = null;
```

Dans l'entity Famille :

```
# [ORM\OneToMany (mappedBy: 'famille', targetEntity: Article::class)]
private Collection $articles;
```

LES FORMULAIRES

Formulaires : les sous formulaires

Si on considère qu'un article a été écrit par une personne

Intégration du formulaire FamilleType dans le formulaire ArticleType:

```
class ArticleType extends AbstractType {  
  
    public function buildForm(FormBuilderInterface $builder, array $options): void {  
        $builder  
            ->add('libelle')  
            ->add('puht')  
            ->add('description')  
            ->add('famille')  
            ->add('famille', FamilleType::class, ['required' => true, 'label' => 'Famille de produit'])  
    }  
}
```



On fera mieux après grâce aux paramètres d'affichage

LES FORMULAIRES

Formulaires : les sous formulaires

Il suffit de construire le formulaire et d'afficher : <http://formulaire50.sym/article/voir/2>


```
#[Route('/voir/{id}', name: 'voir')]
public function voir(int $id, EntityManagerInterface $em): Response
{
    $article=$em->find(Article::class, $id);
    $form=$this->createForm(ArticleType::class, $article);
    return $this->render('article/voir.html.twig', ['formArticle'=>$form->createView()]);
}
```

```
{% extends 'base.html.twig' %}

{% block title %}Client{% endblock %}
{% block body %}
    <h1>Nouveau client</h1>
    {{ form(formArticle) }}
{% endblock %}
```



On fera mieux après grâce aux paramètres d'affichage



← → ↻ formulaire50.sym/article/voir/2

Voir Article

Libelle Samsung QE65QN85B QI

Puissance 1549

Description Couleurs plus riches (Quantum Dots), contenu optimisé en

Famille de produit

Code TV

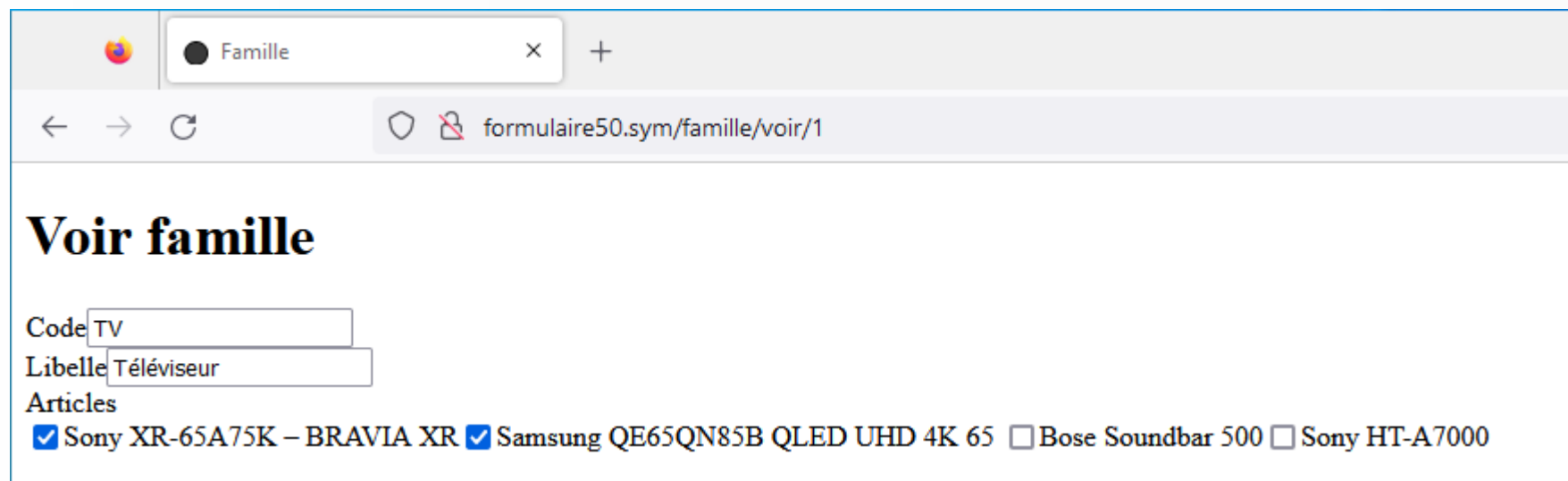
Libelle Téléviseur

LES FORMULAIRES

Formulaires : les sous formulaires

Dans l'autre sens, ça marche aussi ; :

<http://formulaire50.sym/famille/voir/1>



The screenshot shows a web browser window with a single tab titled 'Famille'. The address bar displays 'formulaire50.sym/famille/voir/1'. The page content includes a heading 'Voir famille', a 'Code TV' input field, a 'Libelle Téléviseur' input field, and a section titled 'Articles' containing four items: 'Sony XR-65A75K – BRAVIA XR' (checked), 'Samsung QE65QN85B QLED UHD 4K 65' (checked), 'Bose Soundbar 500' (unchecked), and 'Sony HT-A7000' (unchecked).



On fera mieux après grâce aux paramètres d'affichage

LES FORMULAIRES

Formulaires : mise en forme

On laissera à TWIG le soin de créer lui-même les composants d'affichage



Ce qui permet de séparer les couches en ne gérant pas la présentation du formulaire....

On fera mieux après grâce aux paramètres d'affichage
Pour cela on va utiliser les fonctionnalités de twig sur les formulaires :

Élément TWIG	Action
<code>{{ form_start(nomForm) }}</code>	Début du formulaire
<code>{{ form_row(nomChamp) }}</code>	Création d'un champ
<code>{{ form_end(nomForm) }}</code>	Fin du formulaire



Cette partie sera vue en TD

TRAITEMENT DU FORMULAIRE

LES FORMULAIRES

Traitement du formulaire: ce qu'il faut savoir

Le formulaire soumis revient toujours dans la méthode du contrôleur qui l'a créé

La méthode `handleRequest` de l'objet `Form ($form)` va analyser la requête HTTP

Le contrôleur saura si le formulaire a été retourné ou pas grâce à l'objet `Request`

S'il a été retourné on vérifie :

- S'il a bien été envoyé par l'ordre `submit`
- Qu'il est valide

Si c'est le cas, on gère le retour

On redirige l'application sur une autre route

LES FORMULAIRES

Traitement du formulaire: ce qu'il faut savoir

Pour illustrer le cours nous compléterons la demande de création d'une demande.

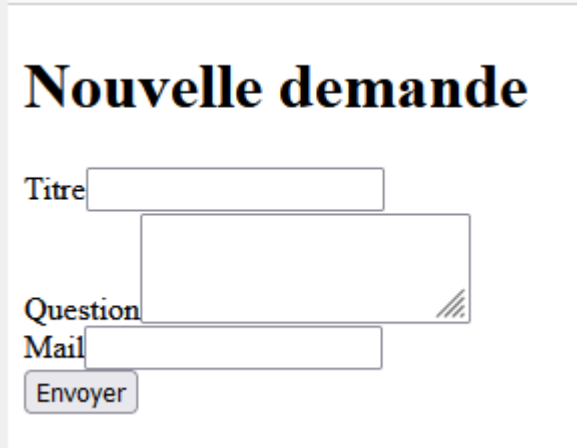
 Le champ `dateCreation` sera renseigné par l'application au retour du formulaire

Il nous faudra :

- rajouter le bouton submit
- récupérer le formulaire dans l'action contrôleur qui l'a affiché
- valider les données saisies
- persister la demande en base de données
- rediriger vers une autre route (home page par exemple)

LES FORMULAIRES

Traitement du formulaire: ce qu'il faut savoir

La classe formulaire :	La vue twig	Le formulaire affiché
	<pre>public function buildForm(FormBuilderInterface \$builder, array \$options) { \$builder ->add('titre') ->add('question') ->add('mail') ; }</pre>	<pre>{% block body %} <h1>Nouvelle demande</h1> {{ form_start(formDemande) }} {{ form_widget(formDemande) }} <input type="submit"/> {{ form_end(form) }} {% endblock %}</pre>

<http://formulaire50.sym/demande/creersubmit>

LES FORMULAIRES

Traitement du formulaire:

La méthode `handleRequest($request)` :

- ✓ récupère les données POST de la requête précédente,
- ✓ les traite
- ✓ les valide (intégrité des données attendues par rapport aux données reçues).

```
public function creerSubmit(Request $request): Response {  
    $demande = new Demande();  
    // On crée le formulaire  
    $form = $this->createForm(DemandeCreerType::class, $demande);  
    $form->handleRequest($request);  
}
```



À partir de cette ligne, si le formulaire a été retourné, les champs du formulaire ont été mappés (bindés) sur les attributs de l'objet.

Traitement du formulaire:

Examen des infos récupérées après la soumission du formulaire :

Nouvelle demande

Titre

Question

Mail

Données renvoyées	Objet \$demande
<div><div>request</div><div><div>parameters</div><div><div>[demande_creer]</div><div>[titre]</div><div>[question]</div><div>[mail]</div></div></div></div> <div>Symfony\Component\HttpFoundation</div> <div>array[1]</div> <div>array[3]</div> <div>string</div> <div>string</div> <div>string</div> <div>...</div> <div>...</div> <div>...</div> <div>"Symfony Formulaires"</div> <div>"Que fait la méthode handleRequest(\$request) ?"</div> <div>"contact@benoitroche.fr"</div>	<div><div>\$demande</div><div><div>titre</div><div>question</div><div>mail</div></div></div> <div>App\Entity\Demande</div> <div>string</div> <div>string</div> <div>string</div> <div>...</div> <div>...</div> <div>...</div> <div>"Symfony Formulaires"</div> <div>"Que fait la méthode handleRequest(\$request) ?"</div> <div>"contact@benoitroche.fr"</div>



Les valeurs du formulaire passées en POST ont strictement été mappées sur les attributs correspondants de l'article \$demande instancié dans le contrôleur

....On va donc pouvoir gérer l'objet \$demande.

LES FORMULAIRES

Traitement du formulaire:

Ensuite il est nécessaire de s'assurer de deux choses :

- ✓ Que le formulaire a bien été soumis : méthode **isSubmitted()** de la classe Form
- ✓ Que le formulaire est valide : méthode **isValid()** de la classe Form

La méthode `isValid()` va demander à Symfony de tester si toutes les contraintes liées aux données sont vérifiées (ex : adresse mail valide, , sexe = M ou F..)

```
$form->handleRequest($request);  
if($form->isSubmitted() && $form->isValid()){  
    // TODO : ici persister l'objet $demande  
}
```

LES FORMULAIRES

Traitement du formulaire:

Il suffit donc maintenant de gérer le retour :

- ✓ Renseigner la date de création
- ✓ Persister la demande saisie
- ✓ Rediriger l'application

```
$form->handleRequest($request);  
if($form->isSubmitted() && $form->isValid()){  
    $demande->setDatecreation(new \DateTime);  
    $em->persist($demande);  
    $em->flush();  
    return $this->redirectToRoute('home');  
}
```

Demande créée

	id	titre	question	mail	datecreation
▶	1	Symfony Formulaires	Que fait la méthode <code>handleRequest(\$request)</code> ?	contact@benoitroche.fr	2022-12-09 08:52:41
*	NULL	NULL	NULL	NULL	NULL

LES FORMULAIRES

Traitement du formulaire:

Nous verrons dans le TD à venir :



- ✓ Comment utiliser le même formulaire pour créer et éditer/modifier un article,
- ✓ En utilisant les routes multiples,

LA VALIDATION DE FORMULAIRE

LES FORMULAIRES

Validation de formulaire:

[Documentation symfony](#)

Symfony est capable de prendre en charge la validation des champs d'un formulaire

Comme on vient de le voir, les champs d'un formulaire sont mappés sur les attributs d'une classe

On pourra donc imposer un certain nombre de règles de validation au niveau du MODEL (**M**VC) c'est-à-dire de l'entity plutôt que du contrôleur.

On va utiliser pour ça les annotations dans les classes Entity

```
use Symfony\Component\Validator\Constraints as Assert;
```



On **renomme** par convention la classe Constraints pour rappeler les classes Assert notamment pour les tests unitaires. Sert aussi à garder une compatibilité avec les versions antérieures de Symfony

Validation de formulaire:

[Documentation symfony](#)

Types de contraintes fournies :

- ✓ de base : Type, IsNull, NotBlank,...
- ✓ sur les chaînes de caractères : longueur, Email, Regex, Json, url,
- ✓ de comparaison : EqualTo, GreaterThan, Range, Unique,....
- ✓ sur les nombres : Positive, PositiveOrZero, Negative, NegativeOrZero,
- ✓ sur les dates : Date, DateTime, Time, TimeZone
- ✓ Sur les choix : Choice, Language, Locale, Country
- ✓ Financières, sur les fichiers ...

LES FORMULAIRES

Validation de formulaire:[Documentation symfony](#)

Les attributs liés à la contrainte vont dépendre de la contrainte.
Quelques exemples de contraintes de validation :

Length

- Basic Usage
- Options
 - allowEmptyString
 - charset
 - charsetMessage
 - exactMessage
 - groups
 - max
 - maxMessage
 - min
 - minMessage
 - normalizer
 - payload

Email

- Basic Usage
- Options
 - checkHost
 - checkMX
 - groups
 - message
 - mode
 - normalizer
 - payload

NotCompromisedPassword

- Basic Usage
- Available Options
 - groups
 - message
 - payload
 - skipOnError
 - threshold

Vérifier que le mot de passe donné n'a pas été compromis en vérifiant qu'il n'est inclus dans aucune des violations de données publiques suivies par le site haveibeenpwned.com

LES FORMULAIRES

Validation de formulaire:

Exemples

<http://formulaire50.sym/demande/creersubmit>

```
# [ORM\Column (length: 100)]
#[Assert\Length(
    min: 5,
    max: 100,
    minMessage: 'Le titre doit avoir au moins {{ limit }} caractères',
    maxMessage: 'Le titre doit avoir au plus {{ limit }} caractères',
)]
private ?string $titre = null;
```

```
# [ORM\Column (length: 120)]
#[Assert\Email(
    message: "L'adresse de courriel {{ value }} n'est pas valide",
)]
private ?string $mail = null;
```


Validation de formulaire:

<http://formulaire50.sym/demande/creersubmit>

En faisant une inspection du code, on s'aperçoit que les contrôles se feront côté serveur :

Forms

- demande_creer
- titre**
- question
- mail

titre

"Symfony\Component\Form\Extension\Core\Type\TextType"

Default Data

Property	Value
Model Format	same as normalized format
Normalized Format	null
View Format	--

Submitted Data

This form was not submitted.

Passed Options

Option	Passed Value	Resolved Value
attr	[{"maxLength" => 100}]	[{"maxLength" => 100}]
required	true	same as passed value

C'est plus sécurisé !!!



HTML généré :

```
<div>
  <label class="required" for="demande_creer_titre">Titre</label>
  <input id="demande_creer_titre" type="text" name="demande_creer[titre]" required="required" maxlength="100">
</div>

<div>
  <label class="required" for="demande_creer_mail">Mail</label>
  <input id="demande_creer_mail" type="text" name="demande_creer[mail]" required="required" maxlength="120">
</div>
```

LES LIENS DANS UN FORMULAIRE TWIG

LES FORMULAIRES

Les Hyperliens :

[Documentation symfony](#)

Symfony a défini des filtres, fonctions et éléments pour twig permettant notamment de gérer les liens.

- ✓ Vers une autre page : fonction path
- ✓ Vers un autre domaine : fonction uri
- ✓ Vers un fichier css ou js : fonction asset ou webpack Encore



Il est préférable de passer par ces fonctions plutôt que de mettre les liens en dur dans le code pour faciliter la maintenance de l'application dans le cas où les cibles des liens seraient modifiées

LES FORMULAIRES

Les chemins : fonction **path**

Liens vers une page : fonction url renvoie le lien absolu

{{ path(route_name, route_parameters = [], relative = false) }}

```
<p> <a href="{{ path('home_homepage') }}" > Home</a>
<p> <a href="{{ path('article_voir', {'id':'1'}) }}" > voir l'article 1</a>
<p> <a href="{{ path('home_bienvenue', {'nom':'Dupont','ville':'Paris'}) }}"> Message de bienvenue</a>
```

```
<p>
  <a href="/home/homepage">Home</a>
</p>
<p>
  <a href="/article/voir/1">voir l'article 1</a>
</p>
<p>
  <a href="/home/bienvenue/Dupont/Paris">Message de bienvenue</a>
</p>
```



Les liens générés sont relatifs

<http://formulaire50.sym/lien/page>

LES FORMULAIRES

Les chemins : fonction **url**

Liens vers une page : fonction **url** renvoie le chemin absolu

{{ url(route_name, route_parameters = {}) }}

```
<p> <a href="{{ url('home_homepage') }}" > Home</a>
<p> <a href="{{ url('article_voir', {'id': '1'}) }}" > voir l'article 1</a>
<p> <a href="{{ url('home_bienvenue', {'nom': 'Dupont', 'ville': 'Paris'}) }}" > Message de bienvenue</a></p>
<p> <a href="http://moodle.benoitroche.fr"> Moodle de Benoît ROCHE</a></p>
```

```
<p>
  <a href="http://formulaire50.sym/home/homepage">Home</a>
</p>
<p>
  <a href="http://formulaire50.sym/article/voir/1">voir l'article 1</a>
</p>
<p>
  <a href="http://formulaire50.sym/home/bienvenue/Dupont/Paris">Message de bienvenue</a>
</p>
<p>
  <a href="http://moodle.benoitroche.fr">Moodle de Benoît ROCHE</a>
```

Les liens générés sont *absolus*

<http://formulaire50.sym/lien/url>



On pourra se servir de cette fonctionnalité lorsque l'on aura besoin de faire de l'AJAX ou appel à une API

LES FORMULAIRES

Les chemins : fonction **asset**

Lien vers un fichier css ou js : mauvaises pratiques :

```
{% block stylesheets %}  
    <link rel="stylesheet" href="css/testcss.css">  
{% endblock %}
```

```
{% block javascripts %}  
    <script src='javascript/testjs.js'></script>  
{% endblock %}
```



Les chemins sont en dur !

Il suffit qu'on change ces chemins et tout est à reprendre !



La solution : **asset**

L'idée est d'installer le package **asset** :

Il permettra de fournir un chemin relatif à partir du répertoire public

composer require symfony/asset

LES FORMULAIRES

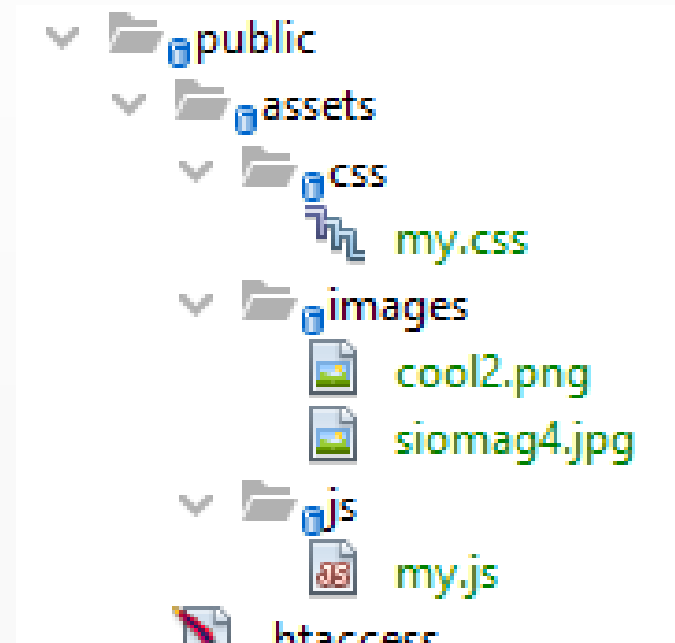
Les chemins : le composant **asset**

Le composant Asset gère :

- ✓ la génération d'URL
- ✓ la gestion des versions

des ressources telles que :

- ✓ les feuilles de style CSS,
- ✓ les fichiers JavaScript
- ✓ les fichiers image.



Le chemin et un chemin RELATIF à partir du dossier public

LES FORMULAIRES

Les chemins : le composant **asset**

On indiquera alors dans le fichier *base.html.twig* ou un autre template le chemin relatif des fichiers par rapport à cette racine :

```
{% block stylesheets %}
    <link rel='stylesheet' href='{{ asset("assets/css/my.css") }}'>
{% endblock %}
{% block javascripts %}
    <script type="text/javascript"
        src="{{ asset('assets/js/my.js') }}"></script>
{% endblock %}

{% block body %}

```

<http://formulaire50.sym/lien/asset>



LES FORMULAIRES

Les chemins : le composant **asset**

Il existe une méthode plus sophistiquée pour gérer les ressources, notamment avec le webpack encore.

<https://symfony.com/doc/5.4/frontend.html>

UN PEU DE STYLE

Les styles : Bootstrap

Il existe 2 grandes façons d'intégrer bootstrap et plus généralement les ressources liées au front dans une application web :

- ✓ En travaillant en mode CDN : c'est le mode le plus largement réparti qui sera privilégié
- ✓ En travaillant en mode local c'est-à-dire en important les fichiers css/js dans l'application..



On peut si on veut utiliser la technologie webpack qui permet de compiler les fichiers de style

LES FORMULAIRES

Les styles : Bootstrap CDN

c'est le mode le plus largement réparti

Il est facile à mettre en œuvre

Il sera privilégié si on n'a pas besoin de personnaliser ou utiliser bootstrap.

CDN (Content Delivery Network):

- ✓ Ensemble de serveurs répartis en de nombreux endroits sur internet
- ✓ Qui répliquent des copies des données. Qui répondent aux requêtes de données en se basant sur les serveurs les plus proches de leurs utilisateurs finaux.
- ✓ Ils rendent les services rapides et moins affectés par les trafics élevés.
- ✓ Ils ne surchargent pas les serveurs d'application qui hébergent les applications



LES FORMULAIRES

Les styles : Bootstrap en CDN

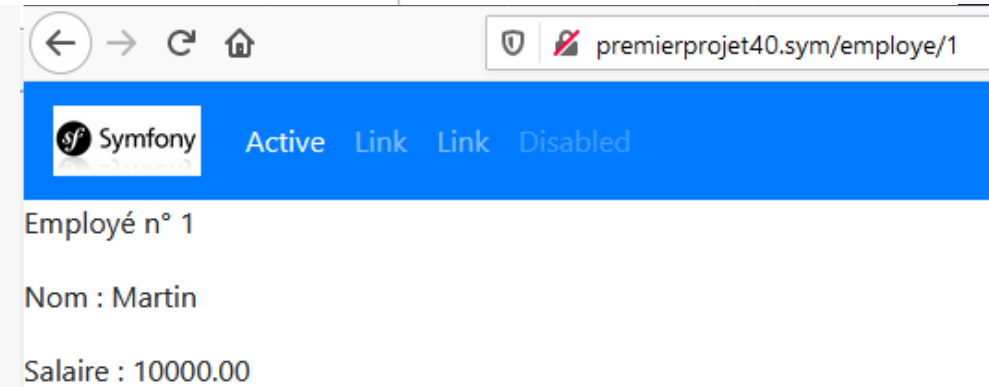
Il suffit de récupérer le lien de la version souhaitée de bootstrap sur le site :
<https://getbootstrap.com/docs/5.2/getting-started/introduction/#cdn-links>

Et de l'inclure dans le formulaire twig de base : base.html.twig :

```
{% block stylesheets %}
    <script rel="https://getbootstrap.com/docs/5.2/getting-started/introduction/#cdn-links">
{% endblock %}

{% block javascripts %}
    <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.2.3/dist/js/bootstrap.bundle.min.js"></script>
{% endblock %}
```

On peut ensuite utiliser un thème trouvé sur internet
Exemple : les sites [W3schools](#) ou [Bootswatch](#) ...



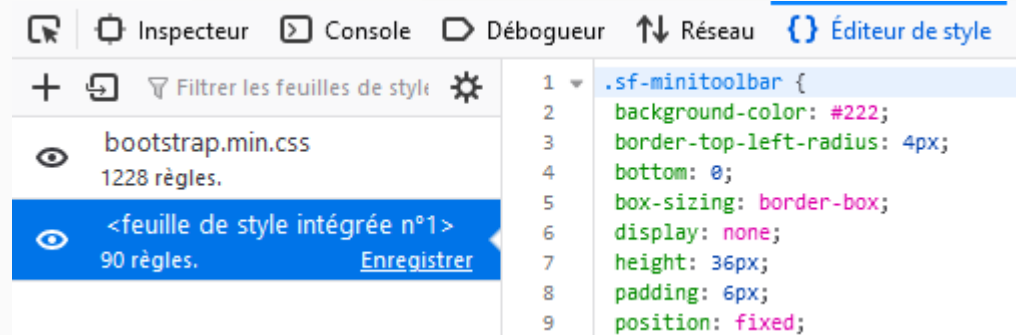
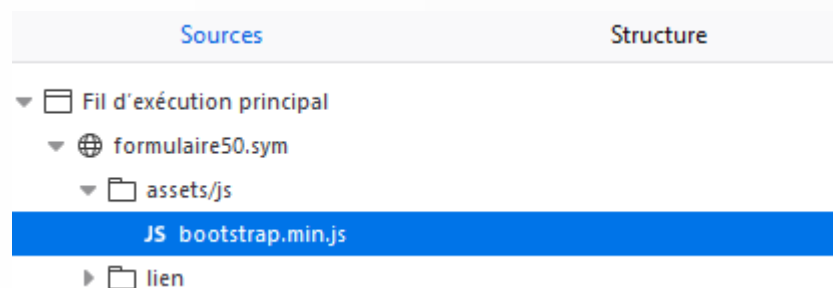
LES FORMULAIRES

Les styles : Bootstrap

On peut aussi tout simplement télécharger et installer bootstrap et l'utiliser avec la fonction twig asset

```
{% block stylesheets %}
    <link rel='stylesheet' href='{{ asset("assets/css/bootstrap.min.css") }}'>
{% endblock %}
{% block javascripts %}
    <script type="text/javascript" src="{{ asset('assets/js/bootstrap.min.js') }}"></script>
{% endblock %}
```

<http://formulaire50.sym/lien/bootstrap>



Les styles : Bootstrap en SASS

Plus compliqué à mettre en œuvre.

- ✓ A privilégier dans le cas d'une intégration continue notamment
- ✓ Il permet de générer un fichier unique des ressources (JS, CSS, images,...)
- ✓ Développé en Ruby, il a été adapté pour nodejs : node-sass.
- ✓ Il nécessite la configuration du loader webpack conçu pour JS à la base mais qui est capable de charger toutes sortes de ressources : css, images,...



Webpack / sass seront abordés en TD dans un module optionnel

EXERCICE

EXERCICE

Il est temps de mettre en pratique tout ceci....



Exercice:

Il est temps de mettre en pratique tout ceci....

[Voir : TD Formulaire](#)



Fin du cours, merci

Questions/Réponses

Benoît Roche
@rocheb83