14 Créer un formulaire avec Symfony :

Nous avions vu avec PHP comment créer des formulaires pour interagir avec une **base de données**. Nous utilisions les super globales **GET** et **POST**.

Le fonctionnement et la mise en œuvre avec Symfony va être différente. Nous allons utiliser les **form_type** (avec **make bundle**).

Les formulaires symfony pourrons fonctionner en GET, POST, PUT, DELETE. Nos formulaire Symfony seront connecté à une entité et pourrons ajouter ou modifier des données.

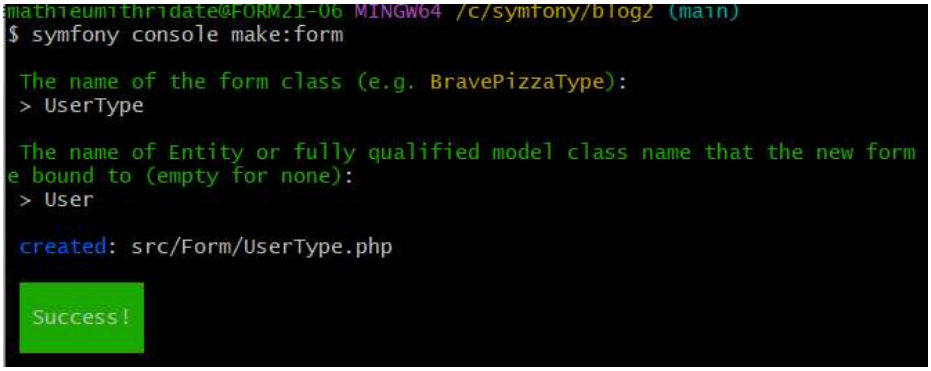
14.1 Générer un formulaire :

Pour générer un formulaire nous allons utiliser la commande suivante (dans un **formulaire**) :

```
symfony console make:form
```

Les formulaires par convention vont se nommer **EntitéForm** :

Dans l'exemple nous allons créer un formulaire **mappé** sur l'entité User avec comme nom **UserType** :



Une nouvelle classe **UserType** va être ajouté au projet dans le dossier **src/form/** :

```
<?php

namespace App\Form;

use App\Entity\User;
use Symfony\Component\Form\AbstractType;
use Symfony\Component\Form\FormBuilderInterface;
use Symfony\Component\OptionsResolver\OptionsResolver;
```

Auteur(s)	Relu, validé et visé par :		Date de création :	Date dernière MAJ :
Mathieu MITHRIDATE	Jérôme CHRETIENNE : **Resp. Secteur Tertiaire & Numérique**		25/01/2021	
	Florence CALMETTES : **Coordinatrice Filière Syst. & Réseaux**			
	Marc CECCALDI **Coordinateur Filière Développement**			

```
class UserType extends AbstractType
{
    public function buildForm(FormBuilderInterface $builder, array $options): void
    {
        $builder
            ->add('name')
            ->add('firstname')
            ->add('mail')
            ->add('password')
            ->add('createdAt')
        ;
    }

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

Dans la méthode **configureOptions** nous allons pouvoir configurer les options de notre formulaire et rattacher la classe (dans notre cas **User::class**).

14.2 Configurer un formulaire ajouter des attributs non mappés :

Nous avons la possibilité d'ajouter des **attributs** supplémentaires à un formulaire qui ne sont pas **mappé** (qui n'existe pas dans **l'entité**) :
Symfony permet 3 paramètres à la fonction **add** :
Le nom de l'attribut, le type et un tableau associatif :
Nous allons ajouter un nouveau champ comme ci-dessous :
//ajout d'un attribut non mappé

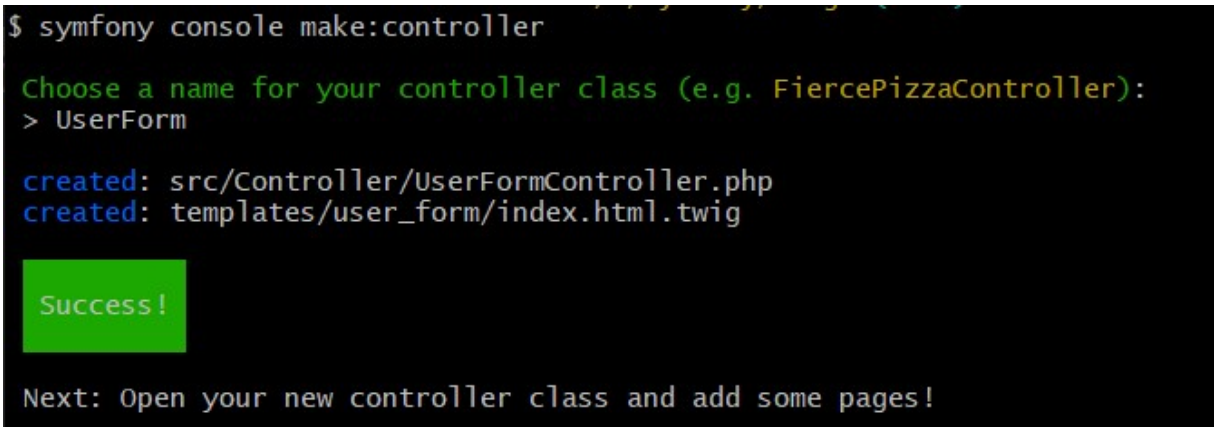
```
->add('age', NumberType::class, [
        'mapped'=>false
    ])
```

Vous retrouverez les différents types de formulaire à l'adresse suivante :
https://symfony.com/doc/2.8/reference/forms/types.html#supported-field-types

14.3 intégrer le formulaire à un controller :

Pour intégrer le formulaire à un controller nous allons utiliser la commande suivante (dans un terminal) :

```
symfony console make:controller
```



Nous allons ensuite éditer la méthode index pour intégrer le formulaire :

```
#[Route('/user/form', name: 'app_user_form')]
    public function index(): Response
    {
        $form = $this->createForm(UserType::class);
        return $this->render('user_form/index.html.twig', [
            'form' => $form->createView()
        ]);
    }
```

Il faut bien penser à ajouter l'import suivant :

```
use App\Form\UserType;
```

14.4 Ajouter le formulaire dans la vue :

Pour ajouter le formulaire nous allons ajouter dans l'interface twig le code ci-dessous :

```
{% block body %}
    {{ form(form) }}
{% endblock %}
```

Auteur(s)	Relu, validé et visé par :		Date de création :	Date dernière MAJ :
Mathieu MITHRIDATE	Jérôme CHRETIENNE : **Resp. Secteur Tertiaire & Numérique**		25/01/2021	
	Florence CALMETTES : **Coordinatrice Filière Syst. & Réseaux**		*Toute reproduction, représentation, diffusion ou rediffusion, totale ou partielle, de ce document ou de son contenu par quelque procédé que ce soit est interdite sans l'autorisation expresse, écrite et préalable de l'ADRAR.*	
	Marc CECCALDI **Coordinateur Filière Développement**			

Si on souhaite ajouter le formulaire champ par champ nous utiliserons le code ci-dessous :

```
{% block body %}
    {{ form_start(form) }}
        {{ form_label(form.name) }}
        {{ form_widget(form.name) }}
        {{ form_rest(form) }}
        <button type="submit">Enregistrer</button>
    {{ form_end(form) }}
{% endblock %}
```

14.5 Ajouter le champ submit :

Pour ajouter le champs submit à un formulaire nous avons 2 méthodes :

Dans la classe form :

```
->add('save', SubmitType::class)
```

Dans l'interface twig :

```
        <button type="submit">Enregistrer</button>
```

14.5 Récupérer les données :

Pour récupérer les données de notre formulaire, nous allons éditer la méthode dans notre controller.

Nous allons ajouter une nouvelle instance de notre classe User dans la méthode **index** et la passer au formulaire :

```
public function index(): Response
{
    $user = new User();

    $form = $this->createForm(UserType::class, $user);

    return $this->render('default/index.html.twig', [
        'form' => $form->createView()
    ]);
}
```

Pour récupérer les données et vérifier que le bouton **submit** nous allons ajouter le code suivant :

```
public function index(Request $request): Response
{

    $user = new User();

    $form = $this->createForm(UserType::class, $user);

    $form->handleRequest($request);
    if ($form->isSubmitted() && $form->isValid()){
        dump($user);die;
    }

    return $this->render('defaultuser_form/index.html.twig', [
        'form' => $form->createView()
    ]);
}
```

14.6 Sauvegarder les données en BDD :

Nous allons maintenant ajouter notre utilisateur en BDD et pour cela nous allons persister les **données** dans notre **entityManager** en utilisant le code ci-dessous :

```
namespace App\Controller;
use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\HttpFoundation\Request;
use Symfony\Component\Routing\Annotation\Route;
use App\Form\UserType;
use App\Entity\User;
use App\Repository\UserRepository;
use Doctrine\ORM\EntityManagerInterface;
public function index(Request $request, EntityManagerInterface
$entityManager): Response
{
    $user = new User();

    $form = $this->createForm(UserType::class, $user);
    $form->handleRequest($request);

    if ($form->isSubmitted() && $form->isValid()){
        $entityManager->persist($user);
```

SYMFONY

```
        $entityManager->flush();
      //refresh la page :
      return $this->redirectToRoute('app_user_form');
    }

    return $this->render('user_form/index.html.twig', [
        'form' => $form->createView()
    ]);
}
```

NB : pensez à ajouter **entityManager** à votre controller.

Auteur(s)	Relu, validé et visé par :		Date de création :	Date dernière MAJ :
Mathieu MITHRIDATE	Jérôme CHRETIENNE : **Resp. Secteur Tertiaire & Numérique**		25/01/2021	
	Florence CALMETTES : **Coordinatrice Filière Syst. & Réseaux**		*Toute reproduction, représentation, diffusion ou rediffusion, totale ou partielle, de ce document ou de son contenu par quelque procédé que ce soit est interdite sans l'autorisation expresse, écrite et préalable de l'ADRAR.*	
	Marc CECCALDI **Coordinateur Filière Développement**			