Création des formulaires

Création de EtudiantForm

Les formulaires peuvent être créés grâce à une ligne de commande en se basant sur une entité existante. Nous allons créer le formulaire permettant de saisir les informations d'un étudiant

Il faut d'abord installer le composant permettant la création des formulaires :

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

La commande ci-dessous permet ensuite de créer un formulaire à partir de l'entité etudiant :

```
>php bin/console make:form
```

Le système nous demande alors le nom de la classe qui contiendra l'ensemble des champs à créer (celui ci doit avoir pour nom : nomEntiteType) ainsi que le nom de l'entité servant de support à la construction du formulaire. A chaque propriété de l'entité (sauf id) correspondra un champ de formulaire.

Exemple pour étudiant :

- *The name of the form class :* EtudiantType
- *The name of Entity or fully qualified model class name that the new form will be bound to :* Etudiant

L'exécution de cette commande créé un nouveau dossier Form dans src et un nouveau fichier EtudiantType, très succint.

Création de la vue

Il faut ensuite créer la vue twig (par exemple *templates/etudiant/ajouter.html.twig*) permettant l'affichage du formulaire. Les tags twig ci-dessous permettent de gérer l'affichage des différents éléments du formulaire.

```
{# templates/etudiant/ajouter.html.twig #}
{{ form_start(form) }}
{{ form_widget(form) }}
{{ form_end(form) }}
```

Modification du contrôleur

Le contrôleur doit être modifié pour générer le formulaire grâce à EtudiantForm et renvoyer vers la vue.

```
public function ajouterEtudiant(){

        $etudiant = new etudiant();
        $form = $this->createForm(EtudiantType::class, $etudiant);
                return $this->render('etudiant/ajouter.html.twig', array(
                'form' => $form->createView(), ));
}
```

Amelioration de EtudiantType

le formulaire ne contient actuellement que des champs de type input text. Nous allons l'enrichir :

```
public function buildForm(FormBuilderInterface $builder, array $options)
    {
        $builder
            ->add('nom', TextType::class)
            ->add('prenom', TextType::class)
            ->add('dateNaiss', DateTimeType::class, array('input' =>
'datetime',
                                                          'widget' =>
'single_text',
                                                          'format' =>
'dd/MM/yyyy',  //L'option 'format' doit être supprimée si le format HTML5
des dates est activée (message d'erreur)
                                                          'required' =>
true,
                                                          'label' =>'date de
naissance',
                                                          'placeholder' =>
'jj/mm/aaaa'))
            ->add('ville', TextType::class)
            ->add('numRue', TextType::class)
            ->add('rue', TextType::class)
            ->add('copos', TextType::class)
            ->add('sexe')
            ->add('surnom', TextType::class)
            ->add('maison', EntityType::class, array('class' =>
'App\Entity\Maison','choice_label' => 'nom' ))
            //->add('promotion')
        ->add('enregistrer', SubmitType::class, array('label' => 'Nouvel
étudiant'))
        ;
    }
```

Ne pas oublier les use :

```
use Symfony\Component\Form\Extension\Core\Type\SubmitType;
use Symfony\Component\Form\Extension\Core\Type\TextType;
use Symfony\Component\Form\Extension\Core\Type\DateType;
```

```
use Symfony\Bridge\Doctrine\Form\Type\EntityType;
use Symfony\Component\Form\Extension\Core\Type\DateTimeType;
```

Tester le nouveau formulaire : http://localhost/webstudent/public/etudiant/ajouter

Le formulaire contient maintenant un champ date devant être saisie obligatoirement (propriété required) et au format français, une liste déroulante proposant les différents maisons. Tous les composants html de base peuvent ainsi être crées avec des propriétés spécifiques. Il est aussi possible de créer son propre composant. https://symfony.com/doc/current/forms.html

Soumettre les données

Il faut maintenant envoyer les données en base. La soumission des données est effectuée dans le contrôleur.

```
public function ajouterEtudiant(ManagerRegistry $doctrine,Request $request){
        $etudiant = new etudiant();
    $form = $this->createForm(EtudiantType::class, $etudiant);
    $form->handleRequest($request);

    if ($form->isSubmitted() && $form->isValid()) {

            $etudiant = $form->getData();

            $entityManager = $doctrine->getManager();
            $entityManager->persist($etudiant);
            $entityManager->flush();

        return $this->render('etudiant/consulter.html.twig', ['etudiant' =>
$etudiant,]);
    }
    else
        {
            return $this->render('etudiant/ajouter.html.twig', array('form'
=> $form->createView(),));
    }
}
```

La fonction prend maintenant en paramètre la requête http (composant à ajouter dans les use). Ne pas oublier d'importer EtudiantType

```
use Symfony\Component\HttpFoundation\Request;
use App\Form\EtudiantType
```

Si les données sont validées, les données sont enregistrées en base et la vue de consultation de l'étudiant est retournée. Sinon, le formulaire est renvoyé de nouveau.

La validation des données

Il existe 3 niveaux de contrôles des données saisies dans le formulaire :

Au niveau de EtudiantForm

Le choix du composant de formulaire permet en lui-même de contrôler les données saisies. Un champ créé grâce à un composant IntegerType n'acceptera pas de caractères alphabétiques. Chaque composant dispose en plus de quelques propriétés supplémentaires permettant d'ajouter des contrôles, notamment required, disabled (voir doc).

Au niveau des entités

Symfony dispose d'un composant, le Validator, permettant d'ajouter des contraintes dans les entités, au niveau des annotations de chaque propriété. Nous allons ajouter des contraintes de validation au niveau de l'entité Etudiant. Il faut d'abord ajouter ce composant :

```
>composer require symfony/validator doctrine/annotations
```

La validator propose plusieurs propriétés selon les composants utilisés. Par exemple, pour ajouter des contraintes de saisie du nombre de caractères sur le nom de l'étudiant, il faut ajouter des annotations au dessus de la propriété nom de l'entité Etudiant.

```
     ....annotations précédentes...
     #[Assert\Length(
        min: 2,
        max: 50,
        minMessage: 'Le nom doit comporter au minimum 2 caractères',
        maxMessage: 'Le nom doit comporter au maximum 50 caractères',
    )]
     private $nom;
```

Pour la date, voici un exemple de message d'erreur dans le cas où la date renseignée ne peut pas être supérieure à la date du jour :

```
....annotations précédentes...
    #[Assert\LessThan('today', message='La date ne peut pas être supérieure à aujourd'hui')]
```

Ne pas oublier le use

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

Ces deux “assertions” obligent l'utilisateur à saisir entre 2 et 50 caractères.

Au niveau des vues

Les formulaires de modifications

Ajout de EtudiantModifierType

Le formulaire de modification qui se nommera EtudiantModifierType étant quasiment le même que celui d'ajout, il « héritera » de EtudiantType.

Nous créons donc une méthode getParent permettant de récupérer tous les champs créés dans EtudiantType. Dans la méthode buildForm, nous désactivons le nom de famille de l'étudiant ; celui-ci n'étant pas modifiable. Nous changeons également le nom du bouton de validation. Nouvelle classe EtudiantModifierType dans le dossier Form :

```
class EtudiantModifierType extends AbstractType
{
    public function buildForm(FormBuilderInterface $builder, array $options)
    {
       $builder
            ->add('nom', TextType::class, array('label' => 'nom étudiant',
'disabled'=> true))
            ->add('enregistrer', SubmitType::class, array('label' =>
'Modifier étudiant'))
             ;

    }

    public function getParent(){
      return EtudiantType::class;
    }

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

Modification du contrôleur

La méthode modifierAction de EtudiantController :

- prend donc en paramètre la Request et l'id de l'étudiant
- récupère un objet etudiant hydraté à partir de l'id passé en paramètre
- créé le formulaire autour de cet objet et de EtudiantModifierType
- si les données sont validées, persiste l'étudiant et renvoie un formulaire de consultation des données modifiées.
- Sinon ré-affiche le formulaire

```
public function modifierEtudiant(ManagerRegistry $doctrine, $id, Request
$request){

    //récupération de l'étudiant dont l'id est passé en paramètre
    $etudiant = $doctrine()
        ->getRepository(Etudiant::class)
        ->find($id);

    if (!$etudiant) {
        throw $this->createNotFoundException('Aucun etudiant trouvé avec le
numéro '.$id);
    }
    else
    {
            $form = $this->createForm(EtudiantModifierType::class,
$etudiant);
            $form->handleRequest($request);

            if ($form->isSubmitted() && $form->isValid()) {

                $etudiant = $form->getData();
                $entityManager = $doctrine()->getManager();
                $entityManager->persist($etudiant);
                $entityManager->flush();
                return $this->render('etudiant/consulter.html.twig',
['etudiant' => $etudiant,]);
           }
           else{
                return $this->render('etudiant/ajouter.html.twig',
array('form' => $form->createView(),));
           }
       }
 }
```