Création d'un formulaire

1ère étape : Ajouter

➔ *php bin/console make:form*

Choisir le nom du formulaire et à quelle table celui-ci est relié.

Dans le formulaire qui vient d'être créé , ici AjoutMotType :

```php
use Symfony\Component\Form\Extension\Core\Type\TextType;
use Symfony\Component\Form\Extension\Core\Type\SubmitType;
use Symfony\Bridge\Doctrine\Form\Type\EntityType;
use App\Entity\Categorie;


class AjoutMotType extends AbstractType
{
    public function buildForm(FormBuilderInterface $builder, array $options)
    {
        $builder
            ->add('libelle', TextType::class)
            ->add('categorie', EntityType::class, array("class"=>"App\Entity\Categorie","choice_label"=>"libelle"))
            ->add('ajouter',SubmitType::class)
        ;
    }
```

On ajoute un libelle de type TEXT, et nous l'importons avec un « use »

On ajoute le champ catégorie qui correspondra à une liste déroulante de toutes les catégories disponibles.

Et enfin , on ajoute le bouton de type submit

/ ! \ Ne pas oublier les imports (use …)

Maintenant, on s'occupe du contrôleur :

➔ php bin/console make:controller

Celui-ci s'appellera MotController.

```php
use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Routing\Annotation\Route;
use Symfony\Component\HttpFoundation\Request;
use App\Form\AjoutMotType;
use App\Form\ModifMotType;
use App\Entity\Mot;

class MotController extends AbstractController
{
    /**
     * @Route("/ajoutMot", name="ajoutMot")
     */
    public function ajoutMot(Request $request): Response
    {
        $mot = new Mot();
        $form = $this->createForm(AjoutMotType::class, $mot);

        if ($request->isMethod('POST')) {
            $form->handleRequest($request);
            if ($form->isSubmitted() && $form->isValid()) {
                $em = $this->getDoctrine()->getManager(); // On récupère le gestionnaire des entités
                $em->persist($mot); // Nous enregistrons notre nouveau thème
                $em->flush(); // Nous validons notre ajout
                $this->addFlash('notice', 'Mot inséré'); // Nous préparons le message à afficher à l'utilisateur sur la page où il se rendra
                }
                return $this->redirectToRoute('listeMots'); // Nous redirigeons l'utilisateur sur l'ajout d'un thème après l'insertion.
                }

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

Le contrôleur créé un nouveau mot, puis un formulaire , puis si le formulaire est valide et le bouton est cliqué, alors on valide les données avec un ‘persist’.

Et enfin on redirige vers la liste des mots. Sans oublier de donner le form à la page.

Et voici pour la vue :

templates > mot > ajoutMot.html.twig

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

{% block title %}{{parent()}}Ajout Theme{% endblock %}

{% block contenu %}
{{parent()}}

<div class="container-fluid">
    <div class="row justify-content-center">
        <h1 class="text-center text-secondary p-4">Ajouter un mot</h1>
    </div>
    <div class="row justify-content-center">
        <div class="col-8 bg-secondary p-4 m-0 text-white">
            {{form(form)}}
        </div>
    </div>
</div>

{% endblock %}
```

2eme étape : lister les données

Pour cela , on s’occupe uniquement de la vue et du contrôleur, voici les 2 fichiers en question :

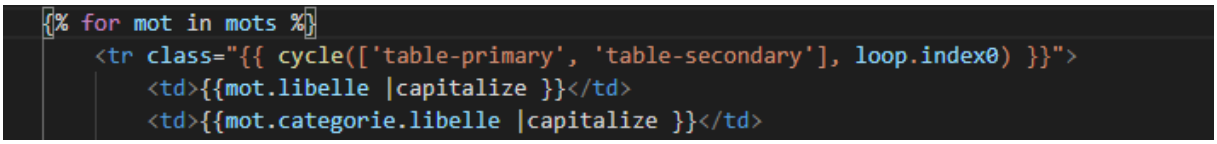
```php
/**
* @Route("/listeMots", name="listeMots")
*/
public function listeMots(Request $request)
{
    $em = $this->getDoctrine();
    $repoMot = $em->getRepository(Mot::class);
    $mots = $repoMot->findBy(array(),array('libelle'=>'ASC'));
    return $this->render('mot/listeMots.html.twig', [
        'mots'=>$mots // Nous passons la liste des thèmes à la vue
    ]);
}
```

```
{% block contenu %}
{{parent()}}
<div class="container-fluid">
    <div class="row justify-content-center">
        <h1 class="text-center text-secondary p-4">Liste des mots</h1>
    </div>
    <div class="row justify-content-center">
        <div class="col-8 p-4 m-0 text-primary">
            <div class="table-responsive">
                <table class="table table-hover">
                    <thead>
                        <tr>
                            <th scope="col">Libellé</th>
                            <th> Categorie </th>
                            <th></th>
                            <th></th>

                        </tr>
                    </thead>
                    <tbody>
                        {% for mot in mots %}
                            <tr class="{{ cycle(['table-primary', 'table-secondary'], loop.index0) }}">
                                <td>{{mot.libelle |capitalize }}</td>
                                <td>{{mot.categorie.libelle |capitalize }}</td>

                                <td><a href="{{path('modifMot',{'id': mot.id})}}" class="text-white"><span class="material-icons" title="Modifier une categorie">create</span></a></t
                                <td><a href="{{path('suppMot',{'id': mot.id})}}" class="text-white"><span class="material-icons" title="Supprimer une categorie"> delete </span></a><
                            </tr>
                        {% endfor %}
                    </tbody>
                </table>
            </div>
        </div>
    </div>
</div>
{% endblock %}
```

Si on zoom sur la partie la plus complexe :

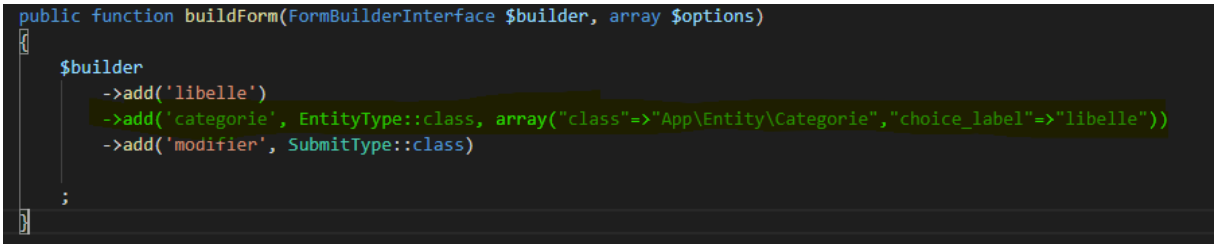


```
{% for mot in mots %}
    <tr class="{{ cycle(['table-primary', 'table-secondary'], loop.index0) }}">
        <td>{{mot.libelle |capitalize }}</td>
        <td>{{mot.categorie.libelle |capitalize }}</td>
```

On voit que pour appeler le nom d'une catégorie qui ne se trouve pas dans la table mot, il suffit d'appeler mot.categorie.(l'objet qui nous intéresse) .

3eme étape : la modification :

On fait un nouveau form avec la même commande que précédemment. Et dedans, voici ce que l'on doit obtenir :



```
public function buildForm(FormBuilderInterface $builder, array $options)
{
    $builder
        ->add('libelle')
        ->add('categorie', EntityType::class, array("class"=>"App\Entity\Categorie","choice_label"=>"libelle"))
        ->add('modifier', SubmitType::class)

    ;
}
```

Ce qui ressemble beaucoup au formulaire d'ajout, à la différence que l'on ne donne pas le type du libelle, et que le bouton est un modifier au lieu d'ajouter.

```php
/**
 * @Route("/modifMot/{id}", name="modifMot", requirements={"id"="\d+"})
 */
public function modifMot(int $id, Request $request): Response
{

    $em = $this->getDoctrine();
    $repoMot = $em->getRepository(Mot::class);
    $mot = $repoMot->find($id);
    if($mot==null){
        $this->addFlash('notice', "Ce mot n'existe pas");
        return $this->redirectToRoute('listeMots');
    }
    $form = $this->createForm(ModifMotType::class,$mot);
    if ($request->isMethod('POST')) {
        $form->handleRequest($request);
        if ($form->isSubmitted() && $form->isValid()) {
            $em = $this->getDoctrine()->getManager();
            $em->persist($mot);
            $em->flush();
            $this->addFlash('notice', 'Mot modifié');
        }
        return $this->redirectToRoute('listeMots');
    }

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

La modification ressemble aussi beaucoup à l'ajout. On utilise « persist » pour modifier comme pour ajouter. Cependant on vérifie que le mot existe bien.

Voici pour la vue :

```twig
{% block title %}{{parent()}}Modif Mot{% endblock %}

{% block contenu %}
{{parent()}}
<div class="container-fluid">
    <div class="row justify-content-center">
        <h1 class="text-center text-secondary p-4">Modifier un mot</h1>
    </div>
    <div class="row justify-content-center">
        <div class="col-8 bg-secondary p-4 m-0 text-white">
            {{form(form)}}
        </div>
    </div>
</div>
{% endblock %}
```

Pour relier la vue à la modification comme la suppression , on utilisera des liens fait à partir d'icone dans la liste des mots , voici le code correspondant :

```
<td><a href="{{path('modifMot',{'id': mot.id})}}" class="text-white"><spa
<td><a href="{{path('suppMot',{'id': mot.id})}}" class="text-white"><span
</tr>
```

4eme étape : La suppression

```
/**
* @Route("/suppMot/{id}", name="suppMot" ,requirements={"id"="\d+"})
*/

public function suppMot(int $id, Request $request): Response
{


    $em = $this->getDoctrine();
    $repoMot = $em->getRepository(Mot::class);
    $mot = $repoMot->find($id) ;

    $em = $this->getDoctrine()->getManager(); // On récupère le gestionn
    $em->remove($mot); // Nous enregistrons notre nouveau thème
    $em->flush(); // Nous validons notre ajout
    $this->addFlash('notice', 'Mot supprimé avec succés'); // Nous prépa


    return $this->redirectToRoute('listeMots');

}
```

Dans le même principe que la modification , mais cette fois ci on utilise « remove » au lieu de « persist ».