GUIDE D'INSTALLATION
CRUD TAG

Introduction :

Bonjour à tous, nous allons créer un CRUD afin de gérer la liste des Tags.

Étape n°1 :

Après la configuration de la base de données sur votre projet, la table « TAG » est déjà créé donc il nous reste plus qu'a installer le bundle « Maker »

```
composer require --dev symfony/maker-bundle
```

Étape n°2 :

Le bundle « Maker » contient une fonctionnalité qui permet de créer tout les fichiers correspondant au CRUD (Il fait un peu le boulot a votre place). Voici la commande à exécuter :

```
symfony console make:crud
```

Après il nous demande sur quelle Entity, on veut créer le CRUD. On choisit donc l'entity « TAG » puis le nom du Controller. On laisse le nom qu'il propose, c'est-à-dire « TagController ».

Étape n°2 BIS (En Option) :

On a redéfini les noms de route afin de les uniformiser avec tout le site complet.

On a personnalisé et internationalisé tout les textes affichés à l'écran (Bouton, Nom de colonne, Titre, En-tête, etc...).

Voici tout ce que l'on a ajouté dans translates/messages.en.yaml et messages.fr.yaml

FR :

```yaml
button:
  show: Info
  back_to_list: Retour à la liste
  list_tag: Catégories
```

```yaml
tags:
  ctTitle: Créer une Nouvelle Catégorie
  cTabTitle: Nouvelle Catégorie
  lTabTitle: Catégorie
  ltTitle: Liste des Catégories
  utTitle: Modifier la Catégorie
  no_found: Catégorie non trouvée
flash:
  delete_impossible: La suppression est impossible la catégorie est en cours d'utilisation
  delete_complete: La suppression  a réussi
  create_impossible: La tâche existe déjà
  create_complete: Une nouvelle tâche a bien été créé
  update_impossible: La tâche ne peut être modifiée car le nom existe déjà
  update_complete: La tâche a été modifiée
```

EN :

```yaml
button:
  show: Show
  back_to_list: back to list
  list_tag: Tags
```

```yaml
tags:
  ctTitle: Create new Tag
  cTabTitle: New Tag
  lTabTitle: Tag Index
  ltTitle: Tags List
  utTitle: Update Tag
  no_found: Tag not found
flash:
  delete_impossible: Removal impossible the tag is used
  delete_complete: Removal completed
  create_impossible: Task already exists
  create_complete: Creation of new task completed
  update_impossible: The task can't be updated because it already exists
  update_complete: The task is updated
```

Étape n°3 :

On a ajouté dans le fichier /config/packages/security.yaml au niveau du access_control la ligne suivante :

```
    - { path: ^/(%app.supported_locales%)/tag, roles: ROLE_ADMIN }
```

Cette ligne sert à bloquer l'accès aux pages «Tag » à tout les utilisateurs qui n'ont pas le rôle « Admin ».

Étape n°4 :

On a ajouté un bouton sur le fichier templates/task/index.html.twig afin de pouvoir accéder au CRUD Task.

```
<a href="{{path('tag_listing')}}" class="btn btn-primary mx-auto">
    {{'general.button.list_tag'|trans}}
</a>
```

Étape n°5 :

Dans le fichier « TagController.php », fonction « Delete », on a ajouté un try catch qui contient un Flash afin de recuperer l'erreur quand la suppression ne se fait pas car il y a des tâches reliées à ce tag.

```php
public function delete(Request $request, Tag $tag, EntityManagerInterface $entityManager): Response
{
    if ($this->isCsrfTokenValid('delete' . $tag->getId(), $request->request->get('_token'))) {
        try {
            $entityManager->remove($tag);
            $entityManager->flush();
        } catch (Exception $e) {
            // dd($e);
            $this->addFlash(
                'danger',
                'flash.delete_impossible'
            );
            return $this->redirectToRoute('tag_listing', [], Response::HTTP_SEE_OTHER);
        }
    }
    $this->addFlash(
        'success',
        'flash.delete_complete'
    );

    return $this->redirectToRoute('tag_listing', [], Response::HTTP_SEE_OTHER);
}
```

Étape n°6 :

Après l'étape n°5, il faut que l'on récupéré ce Flash afin de l'afficher. C'est pour cela que nous allons modifier le fichier templates/tag/index.html.twig :

```
{% for label, messages in app.flashes %}
    {% for message in messages %}
        <div class="alert alert-{{ label }}">
            {{message|trans}}
        </div>
    {% endfor %}
{% endfor %}
```

On peut aussi rajouter un FLASH pour la validation de la suppression mais cela est en option.

Étape n°7 :

Afin de vérifier les doublons au sein de notre table « Tag » dans la base de données, nous allons ajouter un attribut à la donnée membre afin de la rendre unique.

```
/**
 * @ORM\Column(type="string", length=255, unique=true)
 */
private $name;
```

Étape n°8 :

Maintenant, il faut mettre à jour la base de données en effectuant une migration.

Pour créer la migration, il faut faire une commande dans le terminal :

- `SYMFONY CONSOLE MAKE:MIGRATION`

et pour l'exécuter il faut faire :

- `SYMFONY CONSOLE DOCTRINE:MIGRATIONS:MIGRATE`

Étape n°9 :

Pour finir, on a ajouté un try catch dans la fonction « new » et « edit » du fichier TagController.php afin de récupérer les erreurs et afficher un flash pour dire que c'est impossible car il est déjà présent.

new() :

```php
public function new(Request $request, EntityManagerInterface $entityManager, TagRepository $tagRepository): Response
{
    $tag = new Tag();
    // dd($tag);
    $form = $this->createForm(TagType::class, $tag);
    $form->handleRequest($request);
    $arrayTags = $tagRepository->findAll();

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

        try {
            $entityManager->persist($tag);
            $entityManager->flush();
        } catch (Exception $e) {
            $this->addFlash(
                'danger',
                'flash.create_impossible'
            );
            return $this->redirectToRoute('tag_create', [], Response::HTTP_SEE_OTHER);
        }

        $this->addFlash(
            'success',
            'flash.create_complete'
        );
        return $this->redirectToRoute('tag_listing', [], Response::HTTP_SEE_OTHER);
    }

    return $this->renderForm('tag/new.html.twig', [
        'tag' => $tag,
        'form' => $form,
    ]);
}
```

edit() :

```php
public function edit(Request $request, Tag $tag, EntityManagerInterface $entityManager): Response
{
    $form = $this->createForm(TagType::class, $tag);
    $form->handleRequest($request);

    if ($form->isSubmitted() && $form->isValid()) {
        try {
            $entityManager->flush();
        } catch (Exception $e) {
            $this->addFlash(
                'danger',
                'flash.update_impossible'
            );
            return $this->redirectToRoute('tag_update', [], Response::HTTP_SEE_OTHER);
        }
        $this->addFlash(
            'success',
            'flash.update_completed'
        );

        return $this->redirectToRoute('tag_listing', [], Response::HTTP_SEE_OTHER);
    }

    return $this->renderForm('tag/edit.html.twig', [
        'tag' => $tag,
        'form' => $form,
    ]);
}
```

puis dans les fichiers templates/tag/edit.html.twig et templates/tag/new.html.twig, on a ajouter ce qu'il faut pour afficher le flash :

```twig
{% for label, messages in app.flashes %}
    {% for message in messages %}
        <div class="alert alert-{{ label }}">
            {{message|trans}}
        </div>
    {% endfor %}
{% endfor %}
```

Voilà la fin de ce tuto sur la gestion des Tags dans notre application ToDoList.