TP5 : Les Formulaires

L'objectif principal est de remplir notre BDD à partir d'une vue, qui présente un formulaire, pour créer un concert.

ATTENTION : ceux qui ont crée une table qui se nommerait « show » doit impérativement la changer, c'est un mot dit réservé pour SQL.

1. Création du controller & des vues

- Créer une nouvelle route dans le ConcertController : /concert/admin/{id}, qui se nomme « concert_create », qui renvoie l'entité créée.

```php
/**
 * Crée un nouveau concert
 *
 * @Route("/concert/create", name="concert_create")
 *
 */
public function createConcert(Request $request): Response
{
    $show = new Concert();

    $form = $this->createForm( type: ConcertType::class, $show);

    $form->handleRequest($request);
    if ($form->isSubmitted() && $form->isValid()) {
        // $form->getData() holds the submitted values
        // but, the original `$task` variable has also been updated
        $show = $form->getData();

        // ... perform some action, such as saving the task to the database
        // for example, if Task is a Doctrine entity, save it!
         $entityManager = $this->getDoctrine()->getManager();
         $entityManager->persist($show);
         $entityManager->flush();

        return $this->redirectToRoute( route: 'concert_success');
    }

    return $this->render( view: 'concert/new.html.twig', [
            'form' => $form->createView()
        ]
    );
}
```

- Nous ajoutons au controller différentes étapes :

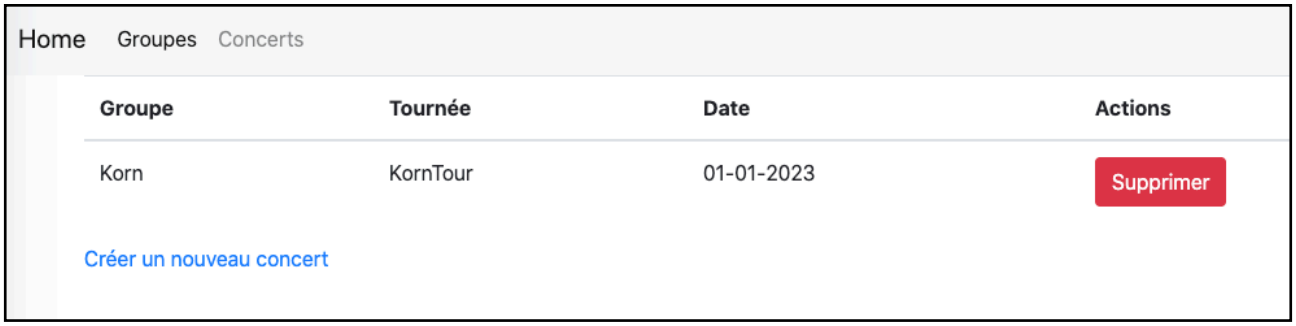
public function createConcert(Request $request) {	La méthode prend en paramètre une request, qui contiendra l'ensemble des données du formulaire.
$concert = new Concert();	On crée l'objet qui récupérera les données du formulaire.
$form = $this->createForm(ConcertType::class, $show);	On crée le form, nommé selon une convention NomClasseType, et on y passe l'entité vide.
$form->handleRequest($request); if ($form->isSubmitted() && $form->isValid()) { $concert = $form->getData();	Si la requête a été envoyé une première fois et que le formulaire est remplie, on vérifie la validité des données envoyées par le formulaire et on récupère les données.
$entityManager = $this->getDoctrine()->getManager(); $entityManager->persist($concert); $entityManager->flush();	On envoie l'entité remplie dans la BDD via Doctrine.
return $this->redirectToRoute('concert_success'); }	Renvoie sur une action spécifique en cas de succès.
return $this->render('concert/new.html.twig', ['form' => $form->createView()]); }	Si la requête était vide et que le formulaire n'était pas rempli, on renvoie le formulaire vide à remplir à la page.

- Ajouter côté template un onglet « Concert » qui contient un lien vers la route en question, au sein d'un texte « Créer un concert ».

```twig
{# show/list.html.twig #}
{% extends 'base.html.twig' %}

{% block body %}
    {% for concert in concerts %}
        <a href="{{ path('band_show', {id: band.id}) }}">{{ concert.band }}</a>
    {% endfor %}
    <div>
        <a href="{{ path('show_admin_create') }}">
        Créer un nouveau concert</a>
    </div>
{% endblock %}
```

Basculer l'affichage des concerts sous forme de tableau grâce à Bootstrap. => Je ne vous indique pas la correction ici, essayer d'afficher un tableau à 4 colonnes comme ceci :



- Ajouter un template correspondant à la nouvelle route pour le create, qui affiche le form.

```twig
{# /show/new.html.twig #}
{% extends 'base.html.twig' %}

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

- Penser à ajouter l'onglet dans la barre Navigateur !

```twig
{# templates/navbar.html.twig #}
<nav class="navbar navbar-expand-lg navbar-light bg-light">
    <a class="navbar-brand" href="/">Home</a>
    <button class="navbar-toggler" type="button" data-toggle="collapse" data-target="#navbarSupportedContent" aria-controls="navbarSu
        <span class="navbar-toggler-icon"></span>
    </button>

    <div class="collapse navbar-collapse" id="navbarSupportedContent">
        <ul class="navbar-nav">
            <li class="nav-item active">
                <a class="nav-link" href="/bands">Groupes</a>
            </li>
            <li class="nav-item">
                <a class="nav-link" href="/shows">Concerts</a>
            </li>
        </ul>
    </div>
</nav>
```

2. Création du ConcertType

- Par la magie de Symfony, un commande permet de créer automatiquement le formulaire correspondant à une entité:

php bin/console make:form

Indiquer le nom de l'entité comme demandé, avec majuscule « Concert » et allez ouvrir le fichier Form/ConcertType crée.

- Que remarquez-vous ?

- Lancez la page d'affichage des concerts, cliquer sur « créer un nouveau concert ». Que se passe-t-il ? Pourquoi?

3. Modification du ConcertType

- Il faut modifier le type du champ « hall » puisqu'il est lié à l'entité « Hall ». D'après la documentation, il faut choisir le EntityType, et indiquer à quelle classe il appartient.

```php
class ConcertType extends AbstractType
{
    public function buildForm(FormBuilderInterface $builder, array $options)
    {
        $builder
            ->add( child: 'date', type: DateType::class, [
                'widget' => 'choice',
                'format' => 'dd / MM / yyyy'
            ])
            ->add( child: 'tourName', type: TextType::class, [
            'label' => 'Nom de la tournée'
            ])
            ->add( child: 'hall', type: EntityType::class,
                ['class' => Hall::class,
                    'choice_label' => 'name'
                ])
```

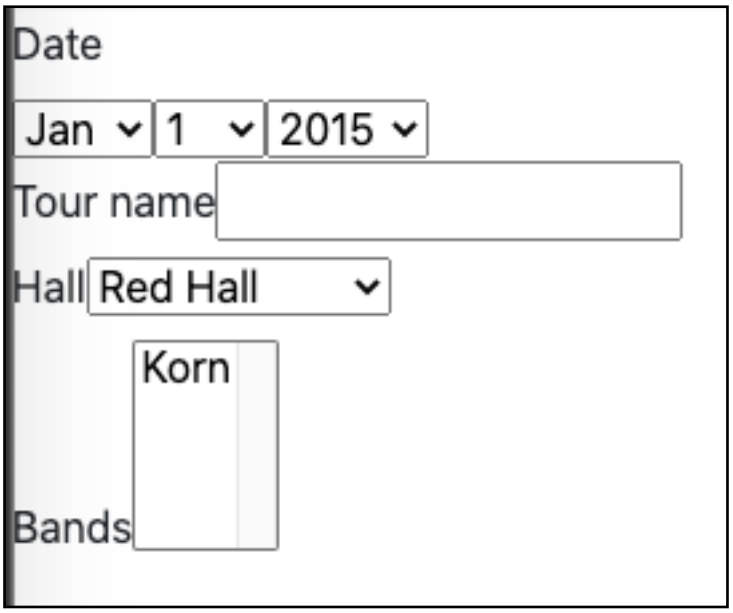
Observer les modifications également sur les champs « date » et « tourName ».

- Modifier en conséquence le champ « bands ».

4. Cliquer sur « Créer un nouveau concert ! »

Et …..

au mieux vous avez ça :



plutôt moche :) Du coup, on fait encore appel à la magie de Bootstrap, on ajoute un

```
{% form_theme form 'bootstrap_4_layout.html.twig' %}
```

dans le template « concert/new.html.twig », sous le « extends ». Et hop !

Tester la création, allez en BDD via PhpMyAdmin ce que ça donne, et revenez sur la page / concert/list pour voir si tout s'affiche correctement :).

Well done !

5. Supprimer un concert

- Nous allons avoir besoin d'une nouvelle route : « /delete/{id} », nommée « concert_delete », à créer dans le ConcertController en annotations.

Elle doit supprimer le concert qui correspond à l'id en BDD.

Pas besoin de créer un template de ce fait, juste de faire l'action dans le controller, et d'ajouter le bouton dans la vue.

```php
/**
 * @param Concert $concert
 *
 * @Route("/delete/{id}", name="concert_delete")
 */
public function delete(Request $request, Concert $concert): Response
{
            $entityManager = $this->getDoctrine()->getManager();
            $entityManager->remove($concert);
            $entityManager->flush();

        return $this->redirectToRoute( route: 'list_concerts');
}
```

On ne « persist() » pas, on « remove() ».

- Pur la vue, créer un bouton qui fait le lien avec cette méthode, vous pouvez encore une fois allez chercher un bouton stylisé bootstrap.
- Plus qu'à tester !

6. Modifier un concert

Modifier une entité, c'est pareil que la créer à l'exception qu'on n'instancie pas une nouvelle entité, on utilise celle qu'on reçoit de la vue.
Du coup : - un bouton « Update » à créer
- une méthode / route : /concert/edit/{id} à créer.
- Vérifier que ça fonctionne !

7. En fait tout peut être automatique… avec des modifications

Vous pouvez faire un :

php bin/console make:crud

CRUD : Create / Read / Update / Delete

Cette commande va générer les méthodes dans le Controller, les Type, les templates correspondants. Sauf qu'il y a de nombreuses modifications à leur apporter à chacun, d'où la nécessité d'avoir bien compris le fonctionnement.

Vous pouvez la jouer sur l'entité « Member » si vous le souhaitez.