TP3: Relations entre entités

Objectif

L'objectif de ce TP est de créer des relations entre entités, ainsi nous pourrons associer les coasters à un parc et une ou plusieurs catégories.

Entité Park

Créez dans un premier temps une nouvelle entité "Park" qui va donc contenir les données des parcs d'attractions.

Je vous rappel (pour la dernière fois) la commande à exécuter:

```
symfony console make:entity Park
```

Entrez les champs suivants:

name	string(80)	not null
country	string(2)	not null
openingYear	integer	null

Le CRUD

Pour faire un CRUD rapidement, il existe encore une commande:

```
symfony console make:crud
```

Sélectionnez ensuite l'entité "*Park*", la commande va créer un CRUD complet avec le contrôleur et le formulaire.

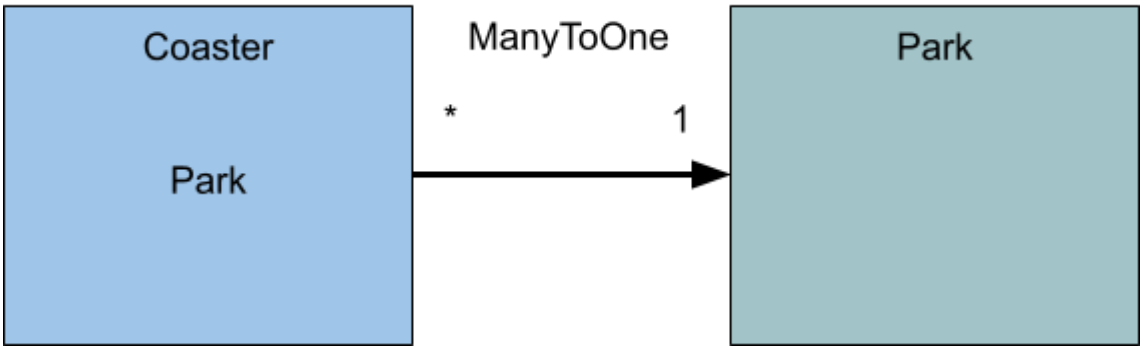
Note: bien sûr il faut modifier un peu de code au niveau des vues pour les adapter à notre design.

⚔️ Défi: dans le formulaire d'édition de parc, modifiez le champ "Country" pour afficher un champ select (il existe un type de champ dans Symfony qui permet celà)

Relation Many To One

Nous allons donc créer une relation ManyToOne pour associer plusieurs coasters pour un parc.

(désolé je n'ai que des références de films des années 80 - 90)



Modifiez l'entité "Coaster" avec la commande *make:entity* en ajoutant un champ *Park* de type *ManyToOne* vers l'entité *Park* (ce champ peut être à null).

⚠️Attention: n'oubliez pas la migration pour mettre à jour la DB.

Si vous êtes en SQLite, la migration ne fonctionnera pas, lancez plutôt cette commande:

```
symfony console doctrine:schema:update --force
```

Champ Park

Pour assigner un parc à un coaster, vous devez ajouter un nouveau champ dans le formulaire d'édition de coaster:

```
->add('park', EntityType::class, [
    'class' => Park::class,
])
```

Le type de champ EntityType va créer automatiquement une requête afin d'afficher tous les parc de la DB.

Si vous essayez d'ajouter ou modifier un coaster, une erreur s'affiche indiquant que Symfony ne sait pas afficher un objet "*Park*", PHP utilise une méthode magique pour afficher un objet sous forme de chaîne.
Ecrivez cette méthode dans l'entité Park:

```
public function __toString(): string
{
    return $this->name;
}
```

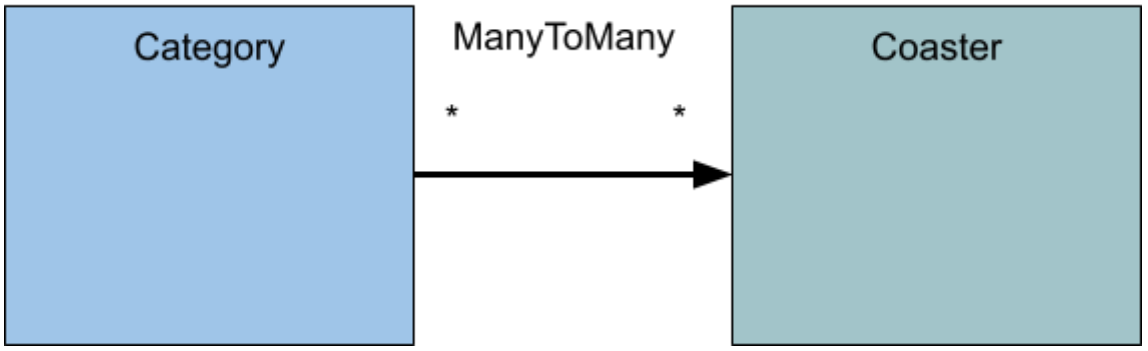
Afficher le parc dans la liste des coasters

Modifiez la vue de la liste des coasters afin d'afficher le parc dans lequel il se trouve.

⚔️ Défi: dans la liste des parcs, afficher le nombre de coasters associés à chaque parc.

Relation ManyToMany

Nous allons maintenant donner des catégories à nos coasters pour permettre de les classer (aquatique, intérieur, bois etc.), il faut donc permettre d'associer plusieurs catégories aux coasters.

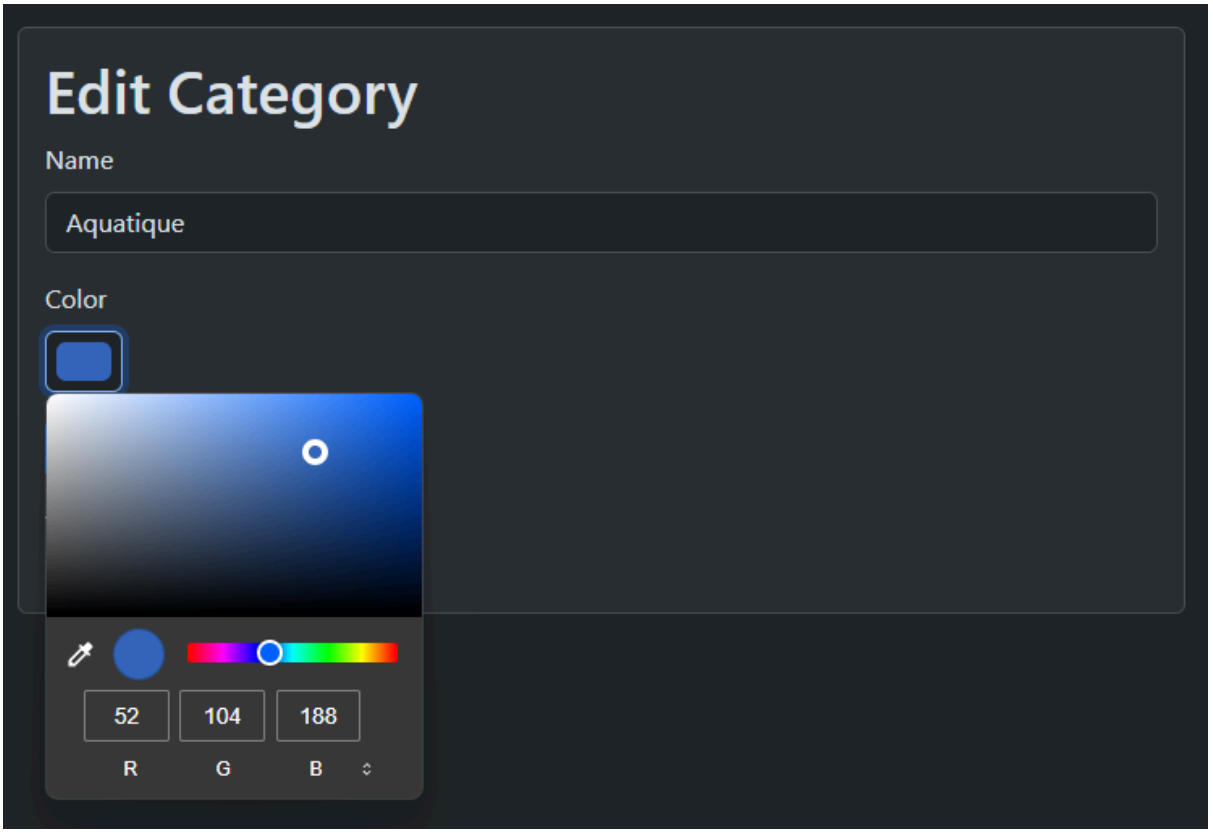


Créer l'entité

Créez une nouvelle entité avec le CRUD:

name	string (80)	not null
color	string(7)	not null

Note: il existe un champ *ColorType*, il faut l'ajouter manuellement dans la classe *CategoryType*, vous pouvez consulter la liste des champs disponibles sur ce lien: https://symfony.com/doc/current/reference/forms/types.html



Ajouter la relation

De la même manière que précédemment, ajouter un champ dans l'entité *Coaster* de type "*ManyToMany*" nommé *Categories*.

⚠️Attention: respectez bien le pluriel, une coaster peut être associée à plusieurs catégories, mettez donc un nom de propriété au pluriel.

Champ catégories

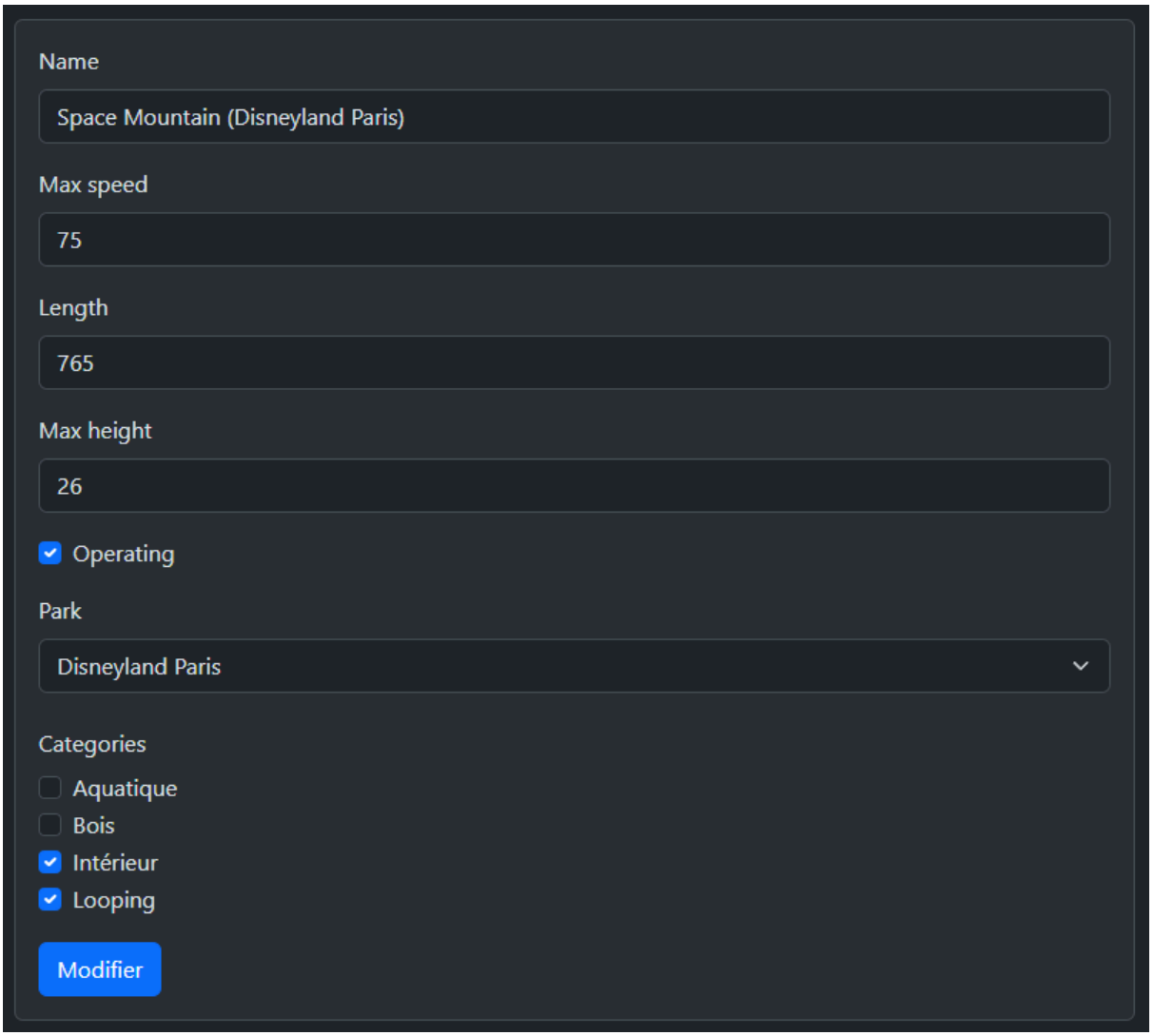
Le champ est également de type *EntityType* mais avec une option ***multiple*** à *true*.

Aidez-vous de la documentation pour ajouter les fonctionnalités suivantes:

- Afficher le champs sous forme de checkboxes
- Afficher la liste par ordre alphabétique (recherchez l'option "*query_builder*")

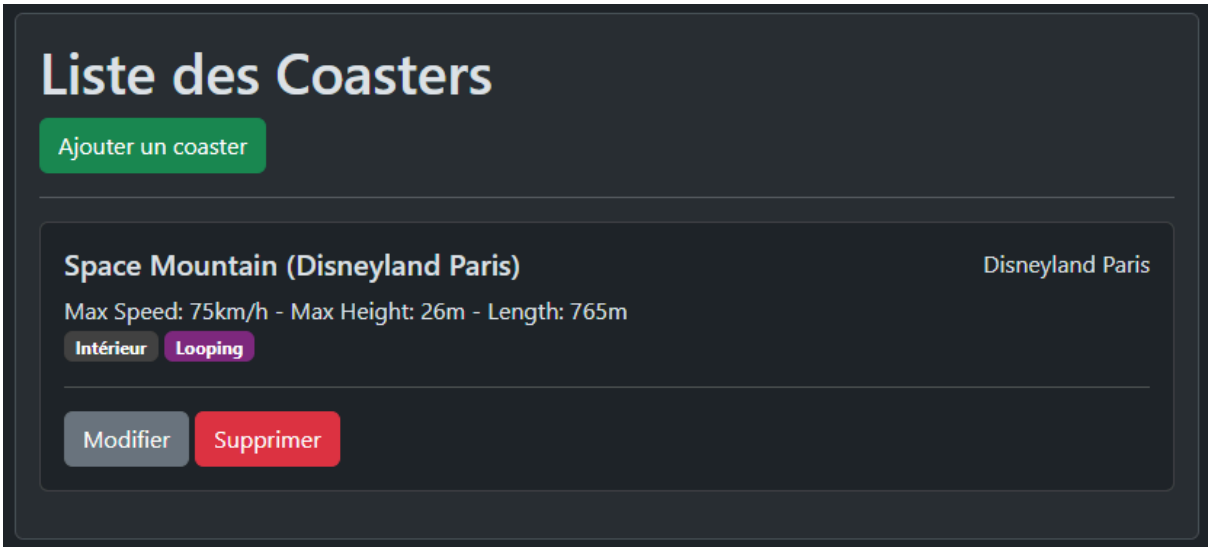
Lien vers la doc:
https://symfony.com/doc/current/reference/forms/types/entity.html

Vous devriez avoir ce formulaire pour les coasters:



Affichage dans la liste

Vous pouvez maintenant afficher les catégories dans la liste des coaster en utilisant une boucle for:



Menu de l'application

Bibliothèque d'icônes

Il existe plusieurs bibliothèques d'icônes tel que FontAwesome, nous allons utiliser les icônes des Bootstrap.

https://icons.getbootstrap.com/

Utiliser npm pour installer la lib:

```
npm i bootstrap-icons --save-dev
```

Puis importez dans le css:

```
@import "bootstrap-icons/font/bootstrap-icons";
```

N'oubliez de recompiler les assets:

```
npm run watch
```

Aidez-vous de la doc DaisyUI pour créer un menu dans le fichier *base.html.twig*:
https://daisyui.com/components/menu/#menu

Filtrer la liste des coasters

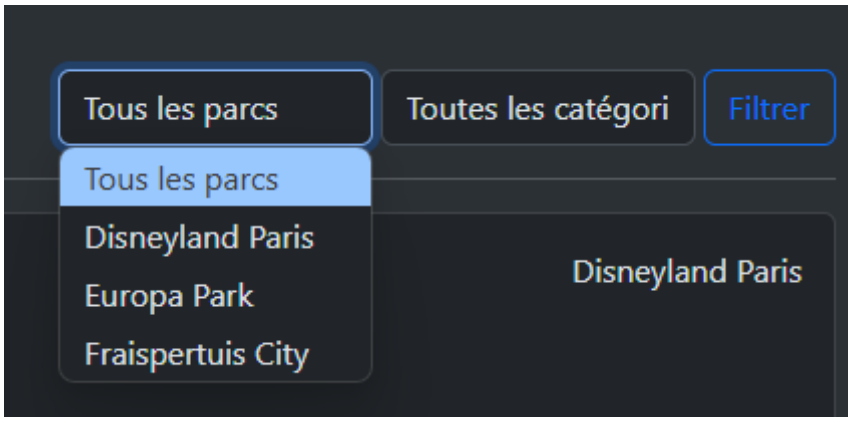
Construction du formulaire

Ici le formulaire est utilisé seulement pour cette page et n'a pas besoin d'être sécurisé car il ne permet pas d'ajouter ou de modifier une entité, il n'est donc pas nécessaire d'utiliser une classe FormType.

Utilisez l'injection de dépendances pour récupérer le ***ParkRepository*** et le ***CategoryRepository*** dans la méthode *index* du *CoasterController*.

Le but est d'envoyer la liste des Parcs et Catégories dans la vue *index* pour les afficher dans un formulaire (GET) sous forme de select.

N'oubliez pas l'attribut Html *name* dans vos champs.



Récupérer les données GET

Dans le contrôleur, l'objet Request va vous permettre de récupérer les données envoyés par le formulaire:

```
$park = $request->get('park', '');
```

Le premier paramètre de la méthode get est le nom de la donnée GET (nom du champ) et le deuxième est la valeur renvoyée si *$_GET['park']* n'est pas trouvée.

Construire la requête

Il reste à créer une nouvelle méthode *findFiltered* dans l'objet *CoasterRepository*.

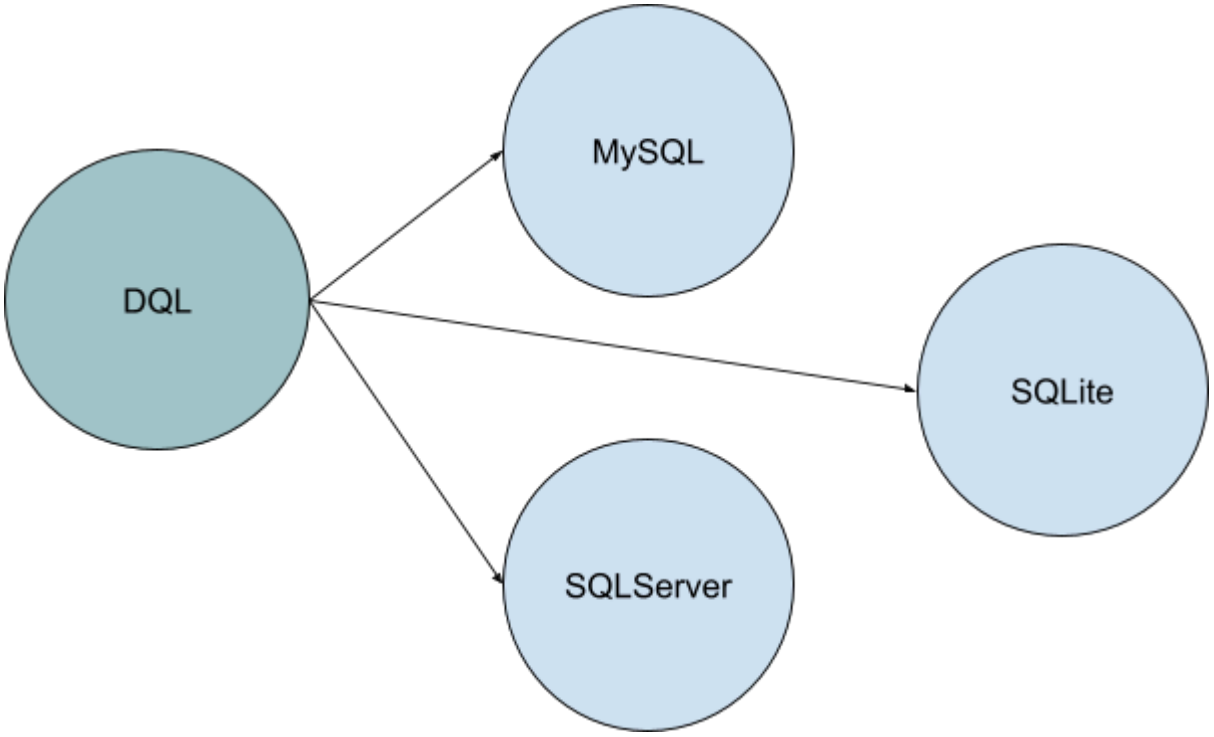
```
public function findFiltered(
    string $parkId = '',
    string $categoryId = '',
): array
{
    $qb = $this->createQueryBuilder('c')
        ->leftJoin('c.park', 'p')
    ;

    if ($parkId !== '') {
        $qb->andWhere('p.id = :parkId')
            ->setParameter('parkId', (int)$parkId)
        ;
    }
    // Filtrer la catégorie

    return $qb->getQuery()->getResult();
```

```
}
```

La méthode *createQueryBuilder* retourne un objet permettant de construire une requête DQL (Doctrine Query Language).



La méthode *leftJoin* va faire une jointure pour récupérer les parcs associés aux coasters, n'oubliez pas de faire la même chose pour les catégories.

Il vous reste à appeler cette méthode dans le *Controller* à la place de *findAll*().

Faire une pagination

Pour faire une pagination nous avons besoin de créer 2 requêtes pour obtenir:

- Les entités de la page courante
- Le nombre total d'entités pour calculer le nombre de pages

Il existe un objet *Paginator* qui fait ces 2 requêtes, modifiez la méthode "*findFiltered*" pour que celle-ci retourne un objet *Paginator* plutôt qu'un array.

```
return new Paginator($qb->getQuery());
```

Vous aurez besoin d'ajouter 2 paramètres dans cette méthode:

- int $count pour le nombre d'élément à afficher
- int $page pour le numéro de la page

Ces deux paramètres vont être utilisés pour définir **l'OFFSET** et la **LIMIT** de la requête.

```
// src/Repository/CoasterRepository.php

$begin = ($page - 1) * $count; // Calcul de l'offset

// ...
$qb = $this->createQueryBuilder('c')
    // ...
    ->setMaxResults($count) // LIMIT
    ->setFirstResult($begin) // OFFSET
```

Calculer le nombre de pages

Dans le contrôleur, il faut calculer le nombre de pages à afficher dans la pagination:

```
$pageCount = max(ceil($coasters->count() / $itemCount), 1);
```

ItemCount est une variable que vous déclarez au début de la méthode pour définir le nombre d'éléments à afficher par page (par exemple 20).

Vous pouvez maintenant modifier l'appel de la fonction findFiltered avec la page courante et le nombre à afficher:

```
$itemCount = 20;
$page = $request->get('p', 1);
```

Afficher la numérotation des pages

Pour afficher une pagination, utilisons un fichier *twig* à part pour pouvoir le réutiliser:

```
{# templates/utils/_pagination.html.twig #}

<nav aria-label="page">
  <ul class="pagination">
    {% for p in 1..pageCount %} {# boucle de 1 à pageCount #}
    <li class="page-item{% if app.request.get('p', 1) == p %}
active{% endif %}">
      <a class="page-link" href="{{ path(route,
app.request.get('_route_params')|merge(app.request.query.all|
merge({'p': p}))) }}">{{ p }}</a>
    </li>
    {% endfor %}
  </ul>
</nav>
```

Insérez maintenant ce fichier dans le vue *index* des coasters:

```
{% include "utils/_pagination.html.twig" with {'route' :
'app_coaster_index'} %}
```

Attention: n'oubliez pas d'envoyer la valeur *pageCount* dans la vue.