Sprint6 Agora:

Sprint6 est divisé en 3 partie :

Dans ce sprint, on est chargé d'apprendre à utiliser le composant de **Symfony**, **l'ORM Doctrine**, en utilisant un extrait des données de l'application fixtures, **les tournois** et leur **catégorie** ainsi que les "**formulaires type**" Doctrine.

I) Demande de Tournoi et Categorie de Tournoi.
II) Associer des participants au tournois.
II) Formulaire Type pour gérer les cat de tournoi

I) Demande de Tournoi et Categorie de Tournoi.

Dans cette itération, on est chargé d'apprendre à utiliser l'un nouveau composant de Symfony, l'ORM Doctrine.

Rappel : Symfony est livré par défaut avec l'ORM Doctrine. Un ORM (pour Object-Relation Mapper, « lien objet-relation ») est un programme qui se place en interface entre un programme applicatif et une base de données relationnelle pour simuler une base de données orientée objet. Il définit des correspondances entre les schémas de la base de données et les classes du programme applicatif.

Les données de la base de données sont des objets. Un objet dont l'enregistrement est confié à l'ORM se nomme une entité

Doctrine utilise les attributs PHP 8 afin de définir la table correspondant à une entité, définir un identifiant (id) en auto-incrément, nommer les colonnes, etc. Ces informations se nomment les metadata de l'entité. Ajouter les metadata à un objet, s'appelle mapper l'objet, c'est-à-dire faire le lien entre l'objet et la représentation physique qu'utilise Doctrine (une table de la base de données relationnelle).

- Crée l'entité Tournoi : php bin/console make:entity

Un fichier src/Entity/Tournoi.php a été créé.
Grâce aux annotations, chaque propriété est mappée (mise en correspondance) avec une colonne de la table de la base de données. Exemple pour la propriété libelle : #[ORM\Column(length: 40)]

```php
<?php

namespace App\Entity;

use App\Repository\TournoiRepository;
use Doctrine\Common\Collections\ArrayCollection;
use Doctrine\Common\Collections\Collection;
use Doctrine\DBAL\Types\Types;
use Doctrine\ORM\Mapping as ORM;

#[ORM\Entity(repositoryClass: TournoiRepository::class)]
class Tournoi
{
    #[ORM\Id]
    #[ORM\GeneratedValue]
    #[ORM\Column]
    private ?int $id = null;

    #[ORM\Column(length: 40)]
    private ?string $libelle = null;

    #[ORM\Column(type: Types::DATETIME_MUTABLE)]
    private ?\DateTimeInterface $date = null;
```

- Ensuite il faut créer la table Tournoi :

Grâce au DoctrineMigrationsBundle il est possible de créer la table à partir de l'entité : php bin/console make:migration

Le fichier "migrations/…. " contiend les ordres SQL.
Pour les exécuter il suffit de passer la commande suivante : php bin/console doctrine:migrations:migrate

- Ensuite on adapte la classe tournoi, Comme la date de création sera toujours égale à la date du jour. Ce comportement sera mis en œuvre dans le constructeur de la classe. On Ajoute le constructeur à la classe Tournoi.

```
public function __construct()
    {
        $this->dateCreation = new \DateTime('now')
```

- Persister les objets dans le contrôleur
  créer un nouveau tournoi et l'enregistrer dans la base de données. On crée l'objet Tournoi et ensuite on persiste l'objet.

```
#[Route('/tournoi/creer', name: 'app_tournoi_creer')]
    public function creerTournoi(EntityManagerInterface $entityManager): Response
    {
        // : Response        type de retour de la méthode creerTournoi
        // pour récupérer le EntityManager (manager d'entités, d'objets)
        //     on peut ajouter l'argument à la méthode comme
ici  creerTournoi(EntityManagerInterface $entityManager)
        //     ou on peut récupérer le EntityManager comme dans la méthode
suivante

        // créer l'objet
        $tournoi = new Tournoi();
        $tournoi->setLibelle('tournoi retrogame 2024');
        $tournoi->setdate(new \DateTime("2024-07-30 00:00:00"));

        // dire à Doctrine que l'objet sera (éventuellement) persisté
        $entityManager->persist($tournoi);

        // exécuter les requêtes (indiquées avec persist) ici il s'agit de l'ordre
INSERT qui sera exécuté
        $entityManager->flush();

        return new Response('Nouveau tournoi enregistré, son id est : ' .
$tournoi->getId());
    }
```

Vérifiez que le serveur MySql tourne : http://localhost:8000/tournoi/creer
Vérifiez le contenu de la base avec la commande suivante. Sous Powershell: php bin/console doctrine:query:sql 'SELECT * FROM tournoi'

- Lire un tournoi de la base de données

```
    $tournoi = $entityManager
    ->getRepository(Tournoi::class)
    ->find($id);
    if (!$tournoi) {
        throw $this->createNotFoundException(
            'Ce tournoi n\'existe pas : ' . $id
        );
    }
```

```
    return new Response('Voici le libellé du tournoi : ' .
$tournoi>getLibelle()); // on peut bien sûr également rendre un
template }
```

Test : http://localhost:8000/tournoi/4

- Modifier , supprimer un tour,oi de la base de données :

```
#[Route('/tournoi/modifier/{id}', name: 'app_tournoi_modifier')]
public function modifier($id, EntityManagerInterface $entityManager)
{
    $tournoi =
$entityManager->getRepository(Tournoi::class)->find($id);
    // en cas de tournoi inexistant, affichage page 404
    if (!$tournoi) {
                    throw $this->createNotFoundException(
                        'Aucun tournoi avec l\'id ' . $id
                    );
            }

            // 2 modification des propriétés
             $tournoi->setLibelle('tournoi RetroGaming milésime 2024');
             // 3 exécution de l'update
              $entityManager->flush();

              // redirection vers l'affichage du tournoi
              return $this->redirectToRoute('app_tournoi_lire', [
                'id' => $tournoi->getId()
            ]);
```

Test : http://localhost:8000/tournoi/modifier/1

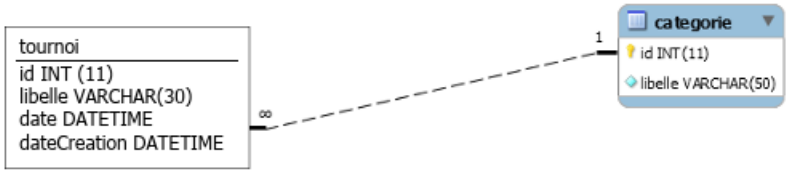
```
#[Route('/tournoi/supprimer/{id}', name: 'app_tournoi_supprimer')]
public function supprimer($id, EntityManagerInterface $entityManager)
{         // 1  recherche du tournoi
        $tournoi =
$entityManager->getRepository(Tournoi::class)->find($id);
                      // en cas de tournoi inexistant, affichage page
404
        if (!$tournoi) {
                throw $this->createNotFoundException(   'Aucun tournoi
avec l\'id ' . $id
);         }          // 2 suppression du tournoi
         $entityManager->remove(($tournoi));         // 3 exécution du
delete
         $entityManager->flush();          // affichage réponse
```

```
                return new Response('Le tournoi a été supprimé, id : '
. $id);      }


                      // en cas de tournoi inexistant, affichage page
404
                      if (!$tournoi) {
                       throw $this->createNotFoundException(    'Aucun
tournoi avec l\'id ' . $id
                    );          }          // 2 suppression du tournoi
                     $entityManager->remove(($tournoi));          // 3
exécution du delete
                     $entityManager->flush();          // affichage
réponse
                     return new Response('Le tournoi a été supprimé, id
: ' . $id);      }
```

- Operation avec deux entités associés

Rappel : Un tournoi appartient à une seule catégorie et une catégorie comprend plusieurs tournois.
Pour Doctrine, ce cas correspond au type d'association : ManyToOne / OneToMany Ce type d'association est courant et mappé avec une clé étrangère. Elle est vue des 2 côtés :
ManyToOne : vue du tournoi
OneToMany : vue de la catégorie



- Créer une association ManyToOne / OneTo Many

Crée entité Catégorie : php bin/console make:entity CatTournois
Nous ajoutons la catégorie au tournoi

```
C:\wamp64\www\agora61>php bin/console make:entity

Class name of the entity to create or update (e.g. GrumpyPuppy):
> Tournoi

Your entity already exists! So let's add some new fields!

New property name (press <return> to stop adding fields):
> categorie

Field type (enter ? to see all types) [string]:
> relation

What class should this entity be related to?:
> CatTournois

What type of relationship is this?
------------ ---------------------------------------------------------------------
 Type         Description
------------ ---------------------------------------------------------------------
 ManyToOne    Each Tournoi relates to (has) one CatTournois.
              Each CatTournois can relate to (can have) many Tournoi objects.

 OneToMany    Each Tournoi can relate to (can have) many CatTournois objects.
              Each CatTournois relates to (has) one Tournoi.

 ManyToMany   Each Tournoi can relate to (can have) many CatTournois objects.
              Each CatTournois can also relate to (can also have) many Tournoi objects.

 OneToOne     Each Tournoi relates to (has) exactly one CatTournois.
              Each CatTournois also relates to (has) exactly one Tournoi.
------------ ---------------------------------------------------------------------

Relation type? [ManyToOne, OneToMany, ManyToMany, OneToOne]:
> ManyToOne

Is the Tournoi.categorie property allowed to be null (nullable)? (yes/no) [yes]:
>

Do you want to add a new property to CatTournois so that you can access/update Tournoi objects from it - e.g.
$catTournois->getTournois()? (yes/no) [yes]:
>
```

public function getCategorie(): ?CatTournois

public function setCategorie(?CatTournois $categorie): static
public function getTournois(): Collection
public function addTournoi(Tournoi $tournoi): static
public function removeTournoi(Tournoi $tournoi): static

migrations : php bin/console make:migration
php bin/console doctrine:migrations:migrate

- Sauvegardez ManyToOne / OneToMany

Modification du contrôleur pour la création des catégories et des tournois

```php
    #[Route('/tournoi/complet/creer', name: 'app_tournoi_complet_creer')]
    public function creerTournoiComplet(EntityManagerInterface $entityManager)
    {
        // créer une catégoriede tournoi
        $categorie = new CatTournois();
        $categorie->setLibelle('Tournois RetroGaming');

        // créer un tournoi
        $tournoi = new Tournoi();
        $tournoi->setLibelle('Tournois e-sport 2024');
        $tournoi->setdate(new \DateTime("2024-10-14 00:00:00"));

        // mettre en relation le tournoi avec la catégorie
        $tournoi->setCategorie($categorie);

        // persister les objets
        $entityManager->persist($categorie);
        $entityManager->persist($tournoi);
        // exécutez les requêtes
        $entityManager->flush();

        // retourner une réponse
        return new Response(
            'Nouveau tournoi enregistré avec l\'id : ' . $tournoi->getId()
                . ' et nouvelle catégorie de tournois enregistrée avec id: ' .
$categorie->getId()
        );
    }
```

Test : http://localhost:8000/tournoi/complet/creer

Pour développer le back office c'est-à-dire l'application web d'administration des données Logma a choisi d'utiliser le framework Symfony.
Il facilitera la prise en charge des évolutions futures déjà envisagées par la MJC Agora.
Dans cette mission vous êtes chargé de compléter l'apprentissage de l'ORM Doctrine en mettant en oeuvre une relation many to many ainsi que les fixtures (les données de tests).

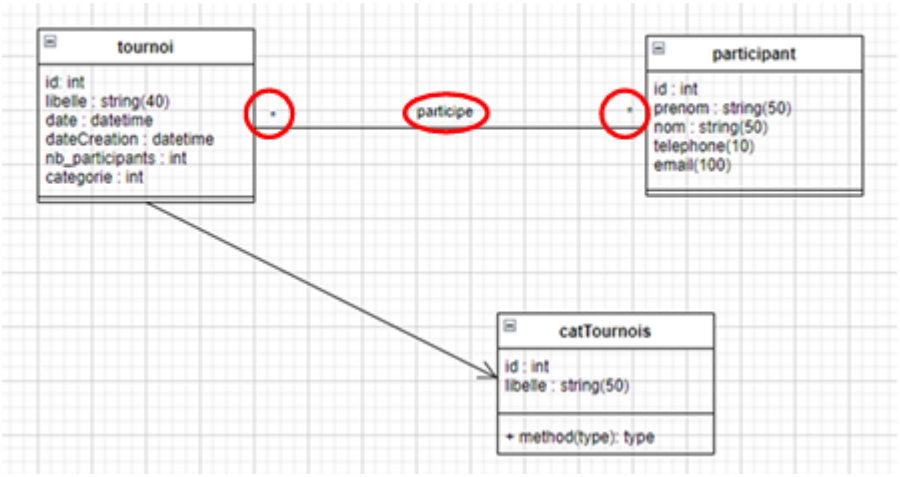
II) Demande 2 : Associer des participants aux tournois

Pour développer le back office c'est-à-dire l'application web d'administration des données Logma a choisi d'utiliser le framework Symfony.
Il facilitera la prise en charge des évolutions futures déjà envisagées par la MJC Agora.
Dans cette mission vous êtes chargé de compléter l'apprentissage de l'ORM Doctrine en mettant en oeuvre une relation many to many ainsi que les fixtures (les données de tests).

- Crée entité Participant : php bin/console make:entity Participant

Il faut créer la relation ManyToMany, pour cela mettons à jour Tournoi

Pour Doctrine ce cas correspond au **type d'association : ManyToMany**




```
C:\wamp64\www\agoram62>php bin/console make:entity Tournoi
 Your entity already exists! So let's add some new fields!

 New property name (press <return> to stop adding fields):
 > particpants

 Field type (enter ? to see all types) [string]:
 > relation

 What class should this entity be related to?:
 > Participant

What type of relationship is this?
 ------------ ---------------------------------------------------------------------
  Type         Description
 ------------ ---------------------------------------------------------------------
  ManyToOne    Each Tournoi relates to (has) one Participant.
               Each Participant can relate to (can have) many Tournoi objects.

  OneToMany    Each Tournoi can relate to (can have) many Participant objects.
               Each Participant relates to (has) one Tournoi.

  ManyToMany   Each Tournoi can relate to (can have) many Participant objects.
               Each Participant can also relate to (can also have) many Tournoi objects.

  OneToOne     Each Tournoi relates to (has) exactly one Participant.
               Each Participant also relates to (has) exactly one Tournoi.
 ------------ ---------------------------------------------------------------------

 Relation type? [ManyToOne, OneToMany, ManyToMany, OneToOne]:
 > ManyToMany

 Do you want to add a new property to Participant so that you can access/update Tournoi objects from it - e.g.
$participant->getTournois()? (yes/no) [yes]:
 >

 A new property will also be added to the Participant class so that you can access the related Tournoi objects from
it.

 New field name inside Participant [tournois]:
 >

 updated: src/Entity/Tournoi.php
 updated: src/Entity/Participant.php
```

Rappel :

L'entité propriétaire doit déclarer l'attribut "inversedBy" : il correspond à la propriété de l'objet "possédé" qui fait le lien entre les deux entités.
L'entité possédée (ou inverse) doit déclarer l'attribut "mappedBy" : il correspond à la propriété de l'objet "propriétaire" qui fait le lien entre les deux entités.

Classe Tournoi

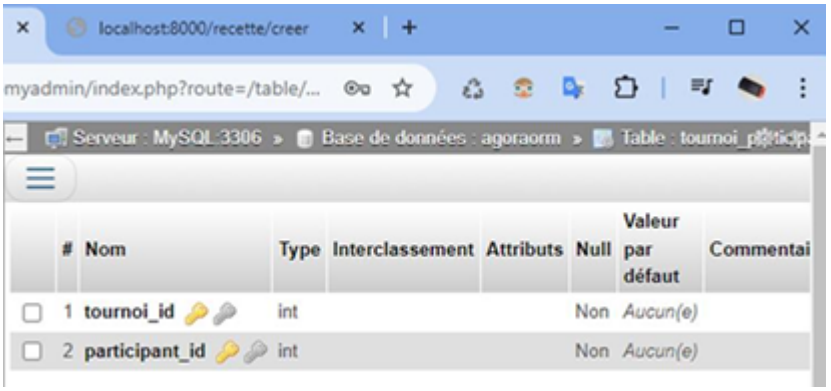
```php
#[ORM\ManyToMany(targetEntity: Participant::class, inversedBy: 'tournois')]
private Collection $particpants;
```

Classe Participant :

```php
#[ORM\ManyToMany(targetEntity: Tournoi::class, mappedBy: 'particpants')]
private Collection $tournois;
```

php bin/console make:migration
php bin/console doctrine:migrations:migrate

La table tournoi_participant a été créée avec 2 clés étrangères, qui sont naturellement les clés primaires des tables tournoi et participant.



rappel : Notez que la représentation de la table tournoi_participant est effectuée par le biais des classes Tournoi ($tournoi) et Participant ($participant) et qu'il n'y a pas d'entité spécifique correspondant à cette table.

- Sauvegarder une association ManyToMany

Ajouter des données dans la table participant :


```php
<?php

use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Routing\Attribute\Route;
use App\Entity\Tournoi;
use App\Entity\CatTournois;
```

```php
use App\Entity\Participant;
use Doctrine\ORM\EntityManagerInterface;

class TournoiController extends AbstractController
{
    #[Route('/tournoi/creer', name: 'app_tournoi_creer')]
    public function creerTournoi(EntityManagerInterface
$entityManager): Response
    {
        // Création d'un nouveau tournoi
        $tournoi = new Tournoi();
        $tournoi->setLibelle('tournoi retrogame 2024');
        $tournoi->setDate(new \DateTime("2024-07-30 00:00:00"));
        $tournoi->setNbParticipants(32);

        // Récupération de la catégorie "RetroGaming"
        $categorie = $entityManager
            ->getRepository(CatTournois::class)
            ->findOneBy(['libelle' => 'RetroGaming']);
        $tournoi->setCategorie($categorie);

        // Liste des prénoms des participants à ajouter
        $prenoms = ['Yannick', 'Marianne', 'Hamid'];

        // Récupération et ajout des participants au tournoi
        foreach ($prenoms as $prenom) {
            $participant = $entityManager
                ->getRepository(Participant::class)
                ->findOneBy(['prenom' => $prenom]);

            if ($participant) {
                $tournoi->addParticipant($participant);
            }
        }

        // Enregistrement en base de données
        $entityManager->persist($tournoi);
        $entityManager->flush();

        // Retour de la réponse
        return new Response('Tournoi créé avec succès.');
    }
}
```

- Créer des données test (fixtures)

rappel : Doctrine fournt une bibliothèque (plus précisément un bundle) nommé DoctrineFixturesBundle qui facilite la mise en place des données de tests ou fixtures.

php bin/console make:fixtures

```
class TournoiFixtures extends Fixture
{
    public function load(ObjectManager $manager): void
    {
        $tbDataTournois = [
            [
                'cattournoi' => 'retrogaming',
                'tournois' => [
                    ['libelle' => 'Retrogame 2024', 'date' => new
\DateTime("2024-08-14 10:00:00"), 'date_creation' => new
\DateTime("2024-05-01 22:31:10"), 'categorie_id' => 1,
'nb_participants' => 64, 'participants' => [
                            ['prenom' => 'Alice', 'nom' => 'Smith',
'telephone' => '1234567890', 'email' => 'alice@example.com'],
                            ['prenom' => 'Bob', 'nom' => 'Jones',
'telephone' => '0987654321', 'email' => 'bob@example.com'],
                        ],
                    ],
                    ['libelle' => 'Retrogame 2023', 'date' => new
\DateTime("2023-06-30 10:00:00"), 'date_creation' => new
\DateTime("2023-04-10 12:42:10"), 'categorie_id' => 1,
'nb_participants' => 34, 'participants' => [
                            ['prenom' => 'Charlie', 'nom' => 'Brown',
'telephone' => '1234567890', 'email' => 'charlie@example.com'],
                            ['prenom' => 'David', 'nom' => 'White',
'telephone' => '0987654321', 'email' => 'david@example.com'],
                        ],
                    ],
                ],
            ],
            [
                'cattournoi' => 'e-sport',
                'tournois' => [
                    ['libelle' => 'Tournois e-sport 2024', 'date' =>
new \DateTime("2024-10-14 10:00:00"), 'date_creation' => new
\DateTime("2024-05-01 22:52:10"), 'categorie_id' => 2,
'nb_participants' => 64, 'participants' => [
                            ['prenom' => 'Eve', 'nom' => 'Black',
'telephone' => '1234567890', 'email' => 'eve@example.com'],
```

```
                            ['prenom' => 'Frank', 'nom' => 'Green',
'telephone' => '0987654321', 'email' => 'frank@example.com'],
                        ],
                    ],
                    ['libelle' => 'Tournois e-sport 2023', 'date' =>
new \DateTime("2023-08-22 10:00:00"), 'date_creation' => new
\DateTime("2023-05-01 12:32:10"), 'categorie_id' => 2,
'nb_participants' => 32, 'participants' => [
                            ['prenom' => 'Grace', 'nom' => 'Adams',
'telephone' => '1234567890', 'email' => 'grace@example.com'],
                            ['prenom' => 'Henry', 'nom' => 'Brown',
'telephone' => '0987654321', 'email' => 'henry@example.com'],
                        ],
                    ],
                    ['libelle' => 'Tournois e-sport 2022', 'date' =>
new \DateTime("2022-07-30 10:00:00"), 'date_creation' => new
\DateTime("2022-05-12 14:22:10"), 'categorie_id' => 2,
'nb_participants' => 32, 'participants' => [
                            ['prenom' => 'Ivy', 'nom' => 'Clark',
'telephone' => '1234567890', 'email' => 'ivy@example.com'],
                            ['prenom' => 'Jack', 'nom' => 'Davis',
'telephone' => '0987654321', 'email' => 'jack@example.com'],
                        ],
                    ],
                ],
            ],
        ];

        for ($i = 0; $i < count($tbDataTournois); ++$i) {
            // Créer une catégorie de tournois
            $categorie = new CatTournois();
            $categorie->setLibelle($tbDataTournois[$i]['cattournoi']);
            $manager->persist($categorie);
```

Lancée les fixtures : php bin/console doctrine:fixtures:load --group=TournoiFixtures --append
Rechargée les fixtures : php bin/console doctrine:fixtures:load

III) Demande 3 : Formulaire type pour gérer les catégorie de tournoi.

Dans cette mission on est chargé de gérer les catégories de tournois en mettant en œuvre les formulaires Symfony

- Crée le controleur CatTournoi php bin/console make:controller CatTournoisController

```
// lire les catégories
        $lesCatTournois = $repository->findAll();
```

```
        return $this->render('cat_tournois/index.html.twig', [
            'formCreation' => $formCreation->createView(),
            'lesCatTournois' => $lesCatTournois,
            'formModification' => $formModificationView,
            'idCatTournoisModif' => $id
        ]);
```

Adaptez la vue c'est-à-dire le template cat_tournois/index.html.twig
{% extends 'base.html.twig' %}
{% block title %}Catégories de tournois{% endblock %}
{% block body %}

- **Créer un "formulaire type" avec Symfony**

Il est possible de créer un formulaire directement dans un contrôleur, mais cette méthode est moins modulaire. Aussi nous allons définir nos formulaires dans des classes, ce qui permettra de réutiliser la classe (le formulaire type) dans n'importe quel contrôleur.
php bin/console make:form

```
C:\wamp64\www\agoram63>php bin/console make:form

The name of the form class (e.g. TinyPuppyType):
> CatTournoisType

The name of Entity or fully qualified model class name that the new form will be bound to (empty for none):
> CatTournois

created: src/Form/CatTournoisType.php
```

- **Afficher le formulaire**

Dans le contrôleur, il faut ensuite créer un formulaire à partir du formulaire type et le passer à la vue

```
 // créer l'objet et le formulaire de création
        $categorie = new CatTournois();
        $formCreation = $this->createForm(CatTournoisType::class,
$categorie);
$formCreation = $this->createForm(CatTournoisType::class,
$categorie);
```

Dans la vue on ajoute une ligne avec le formulaire de création en utilisant les fonctions twig dédiées aux formulaires. En paramètre de la fonction : nom du formulaire ou d'un champ)

```
<!-- formulaire pour ajouter une nouvelle catégorie-->
                    <tr>
                        {{ form_start(formCreation) }}
                        <td class="col-md-1">Nouveau</td>
                        <td class="col-md-6">{{
form_widget(formCreation.libelle) }}
                            {{ form_errors(formCreation.libelle)
}}</td>
                        <td class="col-md-3 ">
```

```
                            <button class="btn btn-primary btn-sm"
type="submit" formaction="{{ path('app_cat_tournois_ajouter') }}"
title="Enregistrer nouvelle catégorie">
                                <i class="fa
fa-save"></i>Enregistrer</button>
                            <button class="btn btn-info btn-sm"
type="reset" title="Effacer la saisie">
                                <i class="fa fa-eraser"></i>
                                Annuler
                            </button>
                        </td>
                        {# Ceci va générer le champ CSRF #}
                        {{ form_rest(formCreation) }}
                        {{ form_end(formCreation) }}
                    </tr>
```

- **Se protéger des attaques CSRF**

"Les attaques CSRF – Cross-Site Request Forgery – sont un vecteur d'attaque puissant et souvent ignoré. Symfony est livré avec les outils pour s'en protéger, mais une approche active est nécessaire pour être totalement protégé." "Symfony propose depuis le départ un outil permettant de protéger les formulaires, les fameux CsrfToken.
Concrètement, il s'agit d'ajouter à la fin de chaque formulaire un token aléatoire, aussi stocké dans la session utilisateur, qui sera validé lorsque l'utilisateur soumet le formulaire. Ce token n'est ni connu ni devinable par un attaquant, il sera donc incapable de fabriquer une fausse requête. Sous conditions que le mécanisme soit bien implémenté, cryptographiquement, c'est une mesure efficace, notre formulaire est protégé. C'est activé par défaut sur tous les formulaires Symfony."

- **Soumettre le formulaire**

Dans le contrôleur on ajoute également une méthode pour enregistrer la nouvelle catégorie de tournois

```
#[Route('/cat/tournois/ajouter', name: 'app_cat_tournois_ajouter')]
    public function ajouter(
        CatTournois $categorie = null,
        Request $request,
        EntityManagerInterface $entityManager,
        CatTournoisRepository $repository
    ) {
        // $categorie objet de la classe CatTournois, il contiendra
les valeurs
        // saisies dans les champs après soumission du formulaire.
        // $request objet avec les informations de la requête HTTP
(GET, POST,
        // ...)
        // $entityManager pour la persistance des données
        // création d'un formulaire de type CatTournoisType
```

```
        $categorie = new CatTournois();
        $form = $this->createForm(CatTournoisType::class,
$categorie);
        // handleRequest met à jour le formulaire
        // si le formulaire a été soumis, handleRequest renseigne
les propriétés
        // avec les données saisies par l'utilisateur et retournées
par la
        // soumission du formulaire
        $form->handleRequest($request);
        if ($form->isSubmitted() && $form->isValid()) {
            // c'est le cas du retour du formulaire
            // l'objet $categorie a été automatiquement "hydraté"
par
            // Doctrine
            // dire à Doctrine que l'objet sera (éventuellement)
persisté
            $entityManager->persist($categorie);
            // exécuter les requêtes (indiquées avec persist) ici
il s'agit de
            // l'ordre INSERT qui sera exécuté
            $entityManager->flush();
            // ajouter un message flash de succès pour informer
l'utilisateur
            $this->addFlash(
                'success',
                'La catégorie de tounois ' .
$categorie->getLibelle() . ' a été ajoutée.'
            );
            // rediriger vers l'affichage des catégories qui
comprend le formulaire
            // pour l"ajout d'une nouvelle catégorie
            return $this->redirectToRoute('app_cat_tournois');
        } else {
            // affichage de la liste des catégories avec le
formulaire de création
            // et ses erreurs
            // lire les catégories
            $lesCatTournois = $repository->findAll();
            // rendre la vue
            return $this->render('cat_tournois/index.html.twig', [
```

```
                'formCreation' => $form->createView(),
                'lesCategories' => $lesCatTournois,
                'formModification' => null,
                'idCategorieModif' => null,
            ]);
        }
```

- **Validation à l'aide des annotations des entités**

```
#[Assert\NotBlank(message: 'Le libellé est obligatoire')]
    #[Assert\Length(
        min: 3,
        max: 50,
        minMessage: 'Le libellé doit comporter au moins {{ limit }}
caractères',
        maxMessage: 'Le libellé ne peut pas dépasser {{ limit }}
caractères',
    )]
```

Dans le template twig on utilise déjà {{ form_errors(formCreation.libelle) }} pour afficher les erreurs

- **Modifier une catégorie de tournois**

```
<!-- formulaire pour modifier une catégorie -->
                                {{ form_start(formModification) }}
                                <td>{{ categorie.id }}</td>
                                <td>{{
form_widget(formModification.libelle) }}

{{form_errors(formModification.libelle) }}</td>
                                <td class="col-md-4">
                                    <button class="btn btn-primary
btn-sm" type="submit" formaction="{{
path('app_cat_tournois_modifier', {'id' : categorie.id }) }}"
title="Enregistrer">
                                        <i class="fa fa-save"></i>
                                        Enregistrer</button>
                                    <button class="btn btn-info
btn-sm" type="reset" title="Effacer la

saisie">
                                        <i class="fa
fa-eraser"></i>
                                        Effacer</button>
```

```
                                    <button class="btn btn-warning
btn-sm" type="submit" formaction="{{ path('app_cat_tournois') }}"
title="Annuler">
                                        <i class="fa fa-undo"></i>
                                        Annuler</button>
                                </td>
                                {# Ceci va générer le champ CSRF #}
                                {{ form_rest(formModification) }}
                                {{ form_end(formModification) }}
                            {% endif %}
                        </tr>
```

Ajoutez une route à la méthode index pour traiter la demande de modification
Ajoutez une méthode pour traiter la modification

- **Supprimer une catégorie de tournois**

dans le contrôleur ajoutez une méthode pour traiter la suppression

```
 #[Route('/ca/tournois/supprimer/{id<\d+>}', name:
'app_cat_tournois_supprimer')]
    public function supprimer(
        CatTournois $categorie = null,
        Request $request,
        EntityManagerInterface $entityManager
    ) {
        // vérifier le token
        if ($this->isCsrfTokenValid('action-item' .
$categorie->getId(), $request->get('_token'))) {
            if ($categorie->getTournois()->count() > 0) {
                $this->addFlash(
                    'error',
                    'Il existe des tournois dans la catégorie ' .
$categorie->getLibelle()
                        . ', elle ne peut pas être supprimée.'
                );
                return $this->redirectToRoute('app_cat_tournois');
            }
            // supprimer la catégorie
            $entityManager->remove($categorie);
            $entityManager->flush();
            $this->addFlash(
                'success',
```

```
                'La catégorie de tournois ' .
$categorie->getLibelle() . ' a été supprimée.'
            );
            return $this->redirectToRoute('app_cat_tournois');
        }
    }
```

Résultat :