Travaux Dirigés et Pratiques de Symfony HAI932I-HAI806I

Michel Meynard

19 décembre 2023

Introduction, paramètres locaux

En cas de problème, veuillez lire les consignes du cours Moodle `https://moodle.umontpellier.fr/course/view.php?id=25137`

Ces TDs et TPs sont destinés à être exécuté dans un environnement comportant :

- un navigateur (client HTTP) type Firefox ou Chrome(ium)
- un serveur HTTP (Apache, Nginx) configuré avec un module interprétant le PHP ; ce serveur peut être local (localhost) ou distant ;
- un serveur MySQL avec un compte d'utilisateur (login, password) et une BD existante sur laquelle l'utilisateur a tous les droits courants ;
- les droits suffisants pour écrire des pages PHP et les transférer (filezilla) ou les déposer sur la machine du serveur HTTP à l'endroit convenable (htdocs, public_html, .. ;) ;
- un outil d'administration phpMyAdmin pour gérer la BD.

La solution la plus simple est d'installer **sur votre machine** un xAMP (LAMP pour Linux, WAMP pour Windows, ...) qui contient un serveur Apache, un serveur MySQL, un module PHP pour Apache et une application web PHPMyAdmin. Une interface graphique permet de lancer les serveurs et d'ouvrir une page Web à la racine des documents Webs (`http://localhost:8888/MAMP/`).

Une autre solution pour un serveur HTTP/PHP servant le répertoire courant consiste à lancer un serveur HTTP localen tapant ce qui suit depuis la ligne de commande puis d'aller visualiser l'URL `http://localhost:8090/mapage.html` :

```
$ php -S localhost:8090
```

1 Installation et création d'un premier projet Symfony

TD/TP 1

Exercice 1 (TD Questions)

1. Que signifie le sigle PHP ?
2. Que signifie le sigle MVC et à quoi sert-il ?
3. Que signifie le mot framework ? Citez-en plusieurs dans le domaine du Web.
4. Que signifient les termes backend, frontend, front-office, back-office dans le domaine du Web ?
5. Qu'est-ce que `composer` dans le monde PHP ? Quels fichiers de config. utilise-t-il ?

Exercice 2 (TP)

Rappelons que `php` est un interpréteur qui peut être utilisée en ligne de commande comme `bash, python` :

- Ouvrez un terminal et lancez-y `php -a` ;
- Taper `<?php` puis tapez des instructions php sans oublier les point-virgules.
- Pour finir, taper `?>` puis CTRL-C pour revenir au bash ;

On peut également exécuter un fichier PHP : `php monfic.php` pour vérifier sa **syntaxe**.

Attention à vérifier la correspondance de version entre l'interprète `/bin/php` et le module PHP d'apache (voir phpinfo()) si vous l'utilisez.

Exercice 3 (TP Installation de composer et de la ligne de commande symfony)

Exécutez les étapes suivantes pour installer composer la ligne de commande Symfony dans votre arborescence et pour pouvoir l'utiliser dans votre projet :

1. S'il n'est pas présent (`which composer`), installer localement composer dans votre répertoire Web (i.e. `public_html`) : `https://getcomposer.org/download/`
2. dans un terminal, depuis votre répertoire d'accueil, téléchargez l'installateur symfony et exécutez-le : `https://symfony.com/download`

   ```
   $ curl https://get.symfony.com/cli/installer | bash
   ```

3. ajoutez le répertoire contenant le binaire exécutable à votre variable d'environnement PATH :

   ```
   $ echo 'export PATH="$HOME/.symfony/bin:$PATH"' >> .bashrc
   ```

4. testez votre nouvelle commande symphony en tapant les commandes suivantes dans des répertoires distincts :

```
$ symfony
$ symfony help
$ symfony serve --help
```

5. rendez-vous dans le répertoire où vous allez stocker vos applications web (i.e. public_html ou htdocs) et créez un nouveau projet (site web complet) :

```
htdocs$  symfony new SfGestionTournoi --webapp
```

6. rendez-vous dans le répertoire SfGestionTournoi et vérifiez sa taille :

```
htdocs$ cd SfGestionTournoi
SfGestionTournoi$ du -sh .
72M     .
```

7. crééz un second projet sans l'option `webapp` et regardez la différence de taille ... Quelle est cette différence ?
8. placez-vous à la racine de votre gros projet et lancez un serveur via la ligne de commande symfony puis ouvrez votre navigateur favori sur la page d'accueil par défaut ... Indiquez les commandes ?
9. votre projet étant complet, il faut :
   (a) Utiliser les caractéristiques du serveur MySQL soit de votre machine que vous administrez, soit de celui fourni par la DSIN dans le portail SAPIENS ...
   (b) pour créer une BD SfGestionTournoi vide dans votre SGBD MySQL (ou MariaDB)
   (c) indiquer dans le fichier `.env` la configuration d'accès à votre BD (qui par défaut est postgreSQL) : pour ma part :
   `DATABASE_URL="mysql://tempo:tempo@127.0.0.1:8889/SfGestionTournoi?serverVersion=5.7"`
10. le bundle Maker étant déjà présent dans ce projet, vous pouvez l'utiliser pour créer un contrôleur de nom Hello.
11. Ouvrez ce contrôleur et ajoutez une route annotée et sa méthode pour afficher "Bonjour Michel" lorsque vous irez sur l'url `/hello/Michel`. On ne vous demande pas de rendu Twig mais juste une réponse textuelle dans Response...
12. Voilà c'est fini

2 Mes premiers contrôleurs

Exercice 4 (TP Factorielle et C_n^p)
On s'intéresse à la fonction factorielle définie par $0! = 1$ et $(n+1)! = (n+1) * n!$. On souhaite écrire un micro-projet Symfony ne contenant qu'une route (/fact/n) permettant d'afficher simplement la valeur de **`n!`**

1. Tapez `symfony -help` afin de visualiser toutes les sous-commandes puis tapez `symfony new -help` pour voir l'aide de la sous-commande new
2. Créer le projet simple (squelette) `SfFactoCombiMastermind` (sans option)
3. Parcourir le répertoire créé. Quel est le contenu du répertoire src/Controller ?
4. Avant l'étape suivante de création d'un contrôleur, il faut installer le bundle **maker** :
   `symfony composer req maker --dev`
5. En Symfony V5), il faut également installler les annotations PHP : `symfony composer req annotations`
6. Lancer la création interactive d'un contrôleur : `php bin/console make:controller` et choisir le nom FactController pour la classe
7. Ouvrir le fichier FactController.php et modifiez-le afin de créer une route annotée /fact/n ... qui retourne simplement n
8. Écrire une fonction PHP `factorielle($n)` externe à la classe contrôleur et qui calcule n ! et retournez cette valeur dans la méthode de route. Vérifiez que 12 !=479001600
9. Créer une nouvelle méthode du même contrôleur permettant de calculer le nombre de combinaisons possibles de n éléments dans p places (triangle de Pascal) grâce à la route `/combi/n/p`. On rappelle la formule $C_n^p = \frac{n!}{p!(n-p)!}$ qui donne par exemple $C_4^2 = 6$

3 Mon premier rendu visuel

Exercice 5 (Twig)
On reste dans le projet précédent mais on souhaite maintenant écrire une page Web sur la route '/' permettant de calculer la fonction factorielle ou bien le nombre de combinaisons avec un rendu HTML. La page devra donc être répartie en 3 blocs :

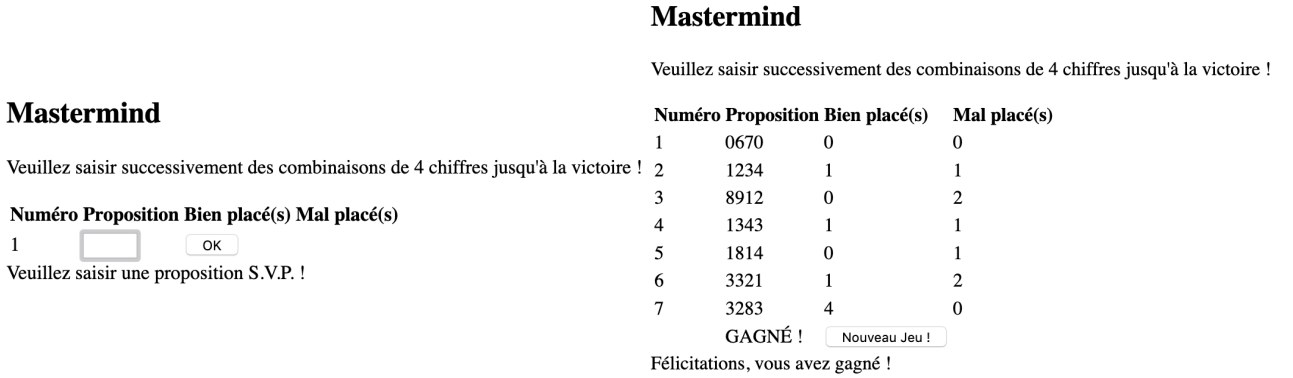
— un formulaire de saisie de n pour le calcul de la factorielle,
— un formulaire distinct de saisie de n et de p pour le calcul des combinaisons,
— un bloc résultat affichant la valeur calculée

Questions

1. Après avoir requis le bundle twig, modifier le fichier `base.html.twig` afin de fixer les deux formulaires constants et de définir un bloc `resultat` variable. La soumission utilisera la méthode GET.
2. Ecrire un un fichier twig (template) factocombi.html.twig héritant du template de base et testant les paramètres reçus (n et r (résultat)) ou (n, p, r) afin de choisir l'affichage concerné (factorielle ou combinaisons).
3. En conservant l'ancien contrôleur, on ajoute une méthode `index` qui récupère les paramètres GET à l'aide de `$request->query->get('n')`, les teste puis retourne le rendu correspondant : template de base si aucun paramètre, sinon le template factocombi avec les bons paramètres !
4. Comment modifier le template de base afin de réafficher les valeurs soumises dans les formulaires lorsqu'on affiche le résultat !

4 Sessions PHP avec symfony

FIGURE 1 – Mastermind



Exercice 6 (Session et Mastermind)
En utilisant la notion de session Symfony, vous créerez un site web permettant de jouer au Mastermind de la figure 1. Le jeu crèe un code aléatoire caché à 4 chiffres différents. Le joueur tente à chaque coup une combinaison de 4 chiffres différents et le jeu lui répond en indiquant le nombre de chiffres bien placés et le nombre de chiffres mal placés. Vous conserverez la liste des coups joués ainsi que leur résultat dans la session Symfony. Vous modéliserez le jeu par une classe PHP qui mémorise le code aléatoire ainsi que la liste des essais successifs et de leurs résultats. Le contrôleur utilisera ce service et le mémorisera dans la session.

1. Comment récupérer le service de session depuis la méthode de contrôleur servant la route "/master" ?
2. Soit l'interface PHP de la classe Mastermind suivante :

```
interface iMastermind extends Serializable { // sauver en session
  public function __construct($taille=4); // crée un nouveau jeu
  public function test($code); // teste une proposition
  public function getEssais(); // ret les prop. préc.
  public function getTaille(); // taille du jeu (4)
  public function isFini(); // vrai si jeu fini : 4 bien placés
}
```

Comment stocker une instance de Mastermind dans une SESSION Symfony ? Qui réalise cette opération ? A quel moment ?

3. Comment le contrôleur récupère-t-il les paramètres soumis par le formulaire ?
4. Écrire la vue du jeu (mastermind.twig)
5. Écrire le contrôleur (fonction master)
6. Programmer la classe Mastermind dans le fichier `Mastermind.php`

5 Base de données et entités Doctrine

Exercice 7 (TP Gestion de tournoi)
Dans le projet complet `SfGestionTournoi` créé lors d'un TP précédent, nous allons créer deux entités `Evenement`, `Tournoi` permettant de modéliser des événements sportifs (i.e. Rolland Garros 2022) et leurs tournois associés (simple messieurs, double dame, ...). Les schémas relationels suivent :

```
Evenement(id, nom varchar(30), dateDeb dateTime, dateFin dateTime)
Tournoi(id, nom varchar(30), description varchar(255), ev_id references Evenement(id)  )
```

1. A l'aide du bundle Maker, créer l'entité `Evenement` contenant un nom requis, une date de début (`dateDeb`) et de fin (`dateFin`) de type datetime et optionnels.
2. Parcourez `src/Entity/Evenement.php` pour vérifier la création de l'entité
3. Même chose pour l'entité `Tournoi` qui possède un `nom` obligatoire de 20 char max, une `description` optionnelle et une propriété `ev` obligatoire de type `ManyToOne` la reliant à une entité `Evenement`. Créez également la propriété inverse (réciproque) `tournois` dans `Evenement` sans activer la contrainte `on delete cascade` (delete orphans : No)
4. En revisitant la classe Evenement, de quoi s'aperçoit-on ?
5. On doit maintenant procéder à la `migration` des entités vers des tables MySQL : relisez l'aide que vous a fournie `make` pour créer le script de migration dans `/migrations`. Quelles commandes SQL sont présentes dans le script ?
6. réalisez la migration puis vérifiez que les 2 tables ont bien été créées dans la BD. Utilisez PHPMyAdmin afin de peupler votre BD avec 2 événements et 4 tournois de votre choix.
7. Après avoir créé quelques événements et tournois dans la BD MySql grâce à PHPMyAdmin, créez un contrôleur TournoiController puis créez une méthode de route `/tournoi` qui affiche la liste des événements et tournois à l'aide de Twig dans un rendu comme suit :

Liste des événements et des tournois

- **1 Rolland Garros**
  - *simple messieurs* ( )
  - *double dames* ( )
- **2 Trophée beach volley 2021**
  - *M15* (réservé aux moins de 15 ans)
  - *Loisir* ( )

Exercice 8 (TP peuplement d'entités)

1. Afin de peupler les événements, créer une route et sa méthode de contrôleur permettant d'ajouter un événement ayant un nom et pas de dates grâce à la route suivante :

```
<?php #[Route("/tournoi/newEvt/{nom<[0-9a-zA-Z ]+>}", name:"newEvt")]
```

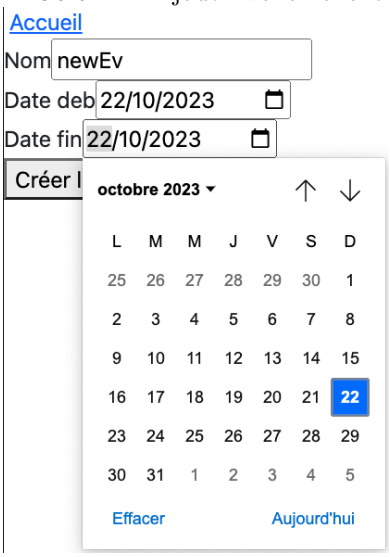
2. Afin de peupler les tournois, créer une route et sa méthode de contrôleur permettant d'ajouter un tournoi à un événement existant. L'id de l'événement et les attributs de tournois seront fournis comme paramètre de la route :

```
<?php #[Route("/tournoi/newTnoi/{evtid<[0-9]+>}/{nom<[0-9a-zA-Z ]+>}/{desc?}", name:"newTnoi")]
```

6 Formulaires

Exercice 9 (TP)
On souhaite créer un formulaire permettant de saisir un nouvel événement dont la vue est donnée dans la figure 2. Pour cela, on réalisera les étapes suivantes :

FIGURE 2 – Ajout Événemenent



1. Créer un type de formulaire `EvType` pour les événements grâce à `make`
2. En plus des trois champs créés automatiquement, ajouter un bouton de soumission dans la méthode `buildForm`
3. Créer un template Twig `Ev.html.twig` qui visualise le plus simplement possible une vue de ce formulaire (nommé form)
4. Ecrire une méthode du contrôleur tournoi `newEv` qui permette de saisir un nouvel événement ET de le rendre persistant
5. Il ne reste plus qu'à peaufiner le rendu en transformant dans `EvType.php` les 2 champs de date afin qu'ils soient au format français (d/m/y) avec un calendrier ...

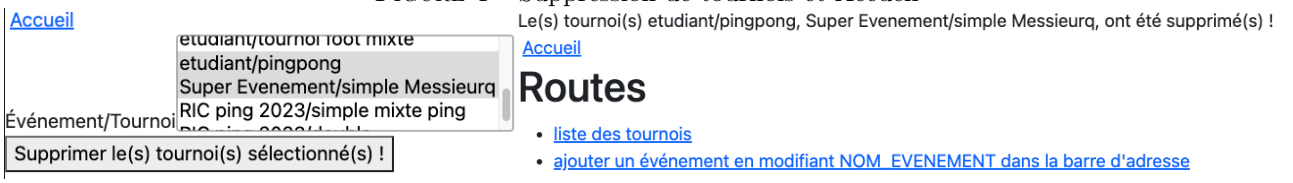
Exercice 10 (TP Formulaire ajout Tournoi)
On souhaite créer un formulaire permettant de saisir un nouveau tournoi dans un événement déjà créé dont la vue est donnée dans la figure 3.

FIGURE 3 – Ajout d'un nouveau tournoi à un événement existant

1. Créer un type de formulaire `TnoiType` pour les tournois grâce à `make`
2. observez le formulaire créé puis créer un template Twig `Tnoi.html.twig` qui visualise le plus simplement possible une vue de ce formulaire (nommé form). Créez une méthode de contrôleur minimale et visualisez ce formulaire ! Que se passe-t-il ?
3. 2 solutions : Utiliser EvType pour visualiser ou saisir un événement, ou bien utiliser le type `EntityType` afin de sélectionner dans le formulaire de tournoi, l'événement existant dans lequel le nouveau tournoi va être inséré. Utilisez la deuxième solution afin d'obtenir le rendu de la figure 3 après avoir modifié le formulaire

Exercice 11 (Projet Gestion de tournois)
On souhaite pouvoir gérer des événements sportifs composés de tournois depuis :
— la phase d'inscription des équipes à un tournoi
— le choix de la formule sportive du premier tour
— le déroulement du tournoi avec la saisie des résultats suivie des formules sportives des tours suivantes
— jusqu'à l'affichage des résultats
On commencera par coder une page de suppression de tournois et la page d'accueil du site comme dans la figure 4.

FIGURE 4 – Suppression de tournois et Accueil



1. créer une page d'accueil composée d'un menu composé de liens permettant de créer un événement (voir exercice 9), de créer un tournoi (voir exercice 10), ...
2. ajouter à cette page d'accueil une division pour afficher un `flashMessage` qui informera l'utilisateur de la réussite ou non de son action précédente.
3. créer un nouveau lien afin de pouvoir supprimer un ou plusieurs tournois existant et coder un formulaire spécifique `supprTnoiType`, un template `supprTnoi.html.twig`, et une méthode de contrôleur.

7 Authentification

Exercice 12 (TP Authentification basique)
On souhaite permettre l'authentification simple par login et mot de passe sur un site de gestion de tournoi (projet webapp déjà créé). On supposera que 3 types d'utilisateurs sont admis :
— l'administrateur du site qui a tous les droits `ROLE_ADMIN`
— les gestionnaires de tournois qui peuvent gérer leurs tournois `ROLE_GEST`
— les utilisateurs anonymes qui peuvent voir les tournois

1. A l'aide du bundle Maker, créer l'entité User avec `login` comme nom unique d'authentification. Préparez la migration et regardez le code SQL dans le script créé.
2. Ajoutez un attribut `evenements` de type `OneToMany` à cette entité afin de pouvoir associer les événements et tournois gérés par les utilisateurs GESTionnaires (associez-lui la propriété réciproque `user`)
3. Dans le vocabulaire Doctrine quelle est l'entité propriétaire et quelle est l'inverse ?
4. A l'aide de PHPMyAdmin et de `security:encode-password`, ajoutez un administrateur ayant le rôle `ROLE_ADMIN` et un gestionnaire du tournoi de Rolland Garros (forget) ainsi qu'un gestionnaire de l'événement de beach volley (beacher)
5. Il ne nous reste plus qu'à créer un système d'authentification (avec formulaire de login, classe Authenticator, SecurityController, route de logout). Puis vous testerez la route `/login` afin de vous authentifier comme beacher : que remarque-t-on ? Testez vos 3 utilisateurs.

Exercice 13 (Migration difficile : restez concentrés !)
Dans l'exercice précédent, nous avons modifié les entités Evenement et User afin de les associer : tout événement à un et un seul gestionnaire. Cependant lors de la migration effective des erreurs surviennent puisque les événements préexistaient par rapport aux utilisateurs :

```
$ sf console doctrine:migrations:migrate
SQLSTATE[23000]: Integrity constraint violation: 1452 Cannot add or update a child row: a foreign key
↪  constraint fails (...FOREIGN KEY (`user_id`) REFERENCES `user` (`id`))
```

En observant la table Evenement, la colonne `user_id` existe bien mais la contrainte d'intégrité n'existe pas ! Le script de migration a été partiellement exécuté : ajout de la colonne OK mais pas l'ajout de la contrainte d'intégrité ni de l'index.

1. Comment faire pour associer chaque événement à son gestionnaire naturel :(Rolland Garros, forget), (Tournoi de beach, beacher) ?
2. Testez à nouveau la migration. Que se passe-t-il ?
3. On va utiliser la commande Doctrine `diff` pour analyser la divergence et proposer un nouveau script de migration :

   ```
   $ sf console doctrine:migrations:diff
   ```

   Visualisez ce nouveau script ! Qu'en pensez-vous ?
4. Le problème, c'est que la commande `migrate` exécute en séquence les migrations succédant à la migration courante (current) jusqu'à la dernière (latest) ! Il vous faut donc exécuter la dernière migration sans exécuter la précédente : cette dernière deviendra la courante et tout ira "mieux" par la suite ! Faites-le après avoir lu l'aide de la commande `execute` :

   ```
   $ sf console doctrine:migrations:execute -help
   ```

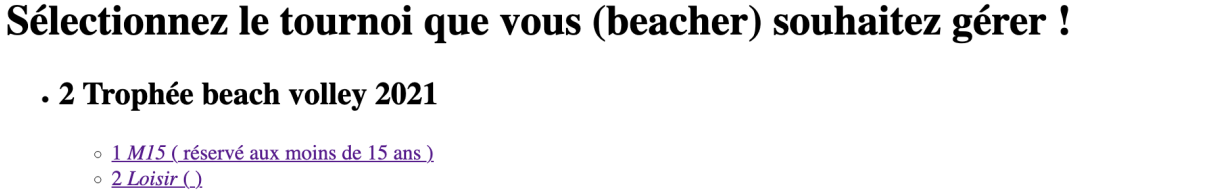
   Attention, le nom de la classe avec son espace de nom doit être fourni à execute :

   ```
   'DoctrineMigrations\Version20180605025653'
   ```
5. Remarquons que l'exécution de migration peut être faite (par défaut ou `-up`) ou défaite (`-down`)

Exercice 14 (Ne voir que les tournois que l'on gère)
On souhaite permettre à chaque gestionnaire de tournoi de ne gérer que ses propres tournois. On crée donc un controleur Gestionnaire avec une route `/gerer` et sa méthode gerer() !

1. créer le contrôleur en l'auto-cablant avec l'interface `UserInterface` pour récupérer l'utilisateur courant
2. Dans l'exercice 7, on avait créer un TournoiController destiné aux anonymes et qui permettaient de voir tous les tournois. On va recopier la méthode du contrôleur en modifiant la requête de sélection de tous les tournois (findAll) par une méthode `findBy(["user"=>$user])`.

3. Afin de permettre la sélection du tournoi courant que le gestionnaire va gérer (inscrire des équipes, constituer un tour avec des poules , ...), on va copier puis modifier le template Twig dans le répertoire `src/templates/gestionnaire` afin de :
   — afficher le nom du gestionnaire courant,
   — transformer chaque nom de tournoi en un lien dirigé vers la route `/gerer/tournoiId`



4. Penser à rediriger vers cette route dans le contrôleur de sécurité après login correct ! Testez des successions de login, logout pour vérifier cette redirection.
5. Pour finir, écrivez une simple (Response) méthode de contrôleur pour gérer un tournoi dont l'id est passé sur la route `gerer/35`. Pensez à vérifier que ce tournoi existe et appartient à l'utilisateur loggé !

6. Contrôle d'accès. L'accès à la route `/gerer/` nécessite d'être loggé car le contrôleur utilise l'auto-cablage sur `UserInterface` ; testez l'accès sans être loggé : vous êtes redirigé sur le formulaire de login. Cependant, on souhaite que seuls les gestionnaires de tournoi puissent accéder aux routes `gerer/*`. Comment modifier la configuration de sécurité ? Testez que l'admin ne puisse gérer les tournois ! On pourrait le permettre en modifiant la hiérarchie des rôles en classant l'admin au-dessus des gestionnaires (il pourra ainsi gérer tous les tournois) :

```
role_hierarchy:
  ROLE_ADMIN: ROLE_GEST
```

7. En ajoutant la directive hiérarchique précéedente, que se passe-t-il lorsque l'admin essaie de gérer ? Modifier le code du contrôleur en conséquence !
8. Ajouter une méthode du contrôleur GestionnaireController associé à la route `/gerer/tid` qui permettra d'afficher un message différent selon que :
   — le tournoi n'existe pas,
   — vous êtes admin ou gestionnaire de ce tournoi et vous pouvez donc gérer ce tournoi tid,
   — vous n'avez pas le droit de gérer ce tournoi,

8 Bundle EasyAdmin (CRUD)

Exercice 15 (Installation et Utilisation de EasyAdmin)
Une interface d'administration des données depuis le site web, réservé à l'administrateur du site, permet d'agir directement sur les données (à la manière de PHPMyAdmin).

1. installer le bundle EasyAdmin dans votre projet de gestion de tournoi
2. créer une interface d'administration soit un tableau de bord sur la route `/admin` géré par le contrôleur `DashboardController`
3. ajouter au tableau de bord un "crud" (interface d'édition) pour les 3 entités Evenement, Tournoi, User
4. modifier la méthode configurant le menu du tableau de bord en ajoutant les 3 crud créés
5. éditer le fichier `TournoiCrudController` afin de permettre à l'administrateur de pouvoir **éditer** :
   — l'événement auquel il est associé (sous la forme d'une liste de sélection)
   — l'id du tournoi (dangereux sauf si `ON UPDATE CASCADE`)

9 API RESTful avec API Platform

Exercice 16 (TD)
On souhaite fournir une API REST pour des événements sportifs composés de tournois.

1. Indiquez les commandes nécessaires pour créer un nouveau projet SfApiTournoi, installer le bundle `api`, lancer un serveur.
2. A l'aide du bundle `maker-bundle`, créez les deux entités Evenement et Tournoi annotées par `@ApiResource()` afin de créer les web services.
3. Jouez avec l'interface graphique en POSTant plusieurs événements sportifs (Rolland-Garros, Tournoi des universités, ...) composés de plusieurs tournois.
4. Continuez à développer l'API tout en construisant un frontend Angular ou autre pour gérer des événements sportifs ...