Ecole Supérieure Privée Technologies & Ingénierie

Type d'épreuve	: ☐ Devoir ■ Examen
Enseignant	: Wissem ELJAOUED
Matière	: Framework PHP (Symfony)
Année Universitaire	: 2024-2025 **Semestre** : 1
Classe	: ING-4-J-GLSI
Documents	: ☐ Autorisés ■ Non autorisés
Date	: **14-01-2025** **Durée** : 1h :30
Nombre de pages	: 8 pages

Exercice 01 : (10 pts)

A. QCM (6 pts)

1- La méthode **path()** prend en argument
   a- Une URL
   b- Un nom de Route
   c- Une page Twig
   d- Aucune de ces réponses n'est vraie.

2- Pour afficher l'attribut ***date_depart*** d'un objet ***$vol*** sous la forme « m/d/y » dans une vue Twig. Quelle est la bonne syntaxe ?
   a- {{ vol.date_depart | date("m/d/y") }}
   b- {{ $vol.date_depart | date("m/d/y") }}
   c- {% $vol.date_depart | date("m/d/y") %}
   d- {{ date(vol.date_depart, "m/d/y") }}

3- Parmi les codes du contrôleur suivant, lequel supprime correctement un **club** identifié par un $id ?

a) $club = $clubRepository-> find($id); $em->remove(); $em->flush();	**c)** $club = $clubRepository-> find($id); $em->remove($club); $em->flush();
b) $club = $clubRepository-> find($id); $em->remove();	**d)** $club = $clubRepository-> find($id); $em->delete(); $em->flush();

4- Dans l'architecture de symfony, quel est le composant qui intercepte les URLs provenant d'un utilisateur :
   a- Controller
   b- FrontController
   c- Router
   d- Kernel

5- Quelle est la commande pour exécuter une migration ?
   a- php bin/console doctrine:migration:execute

b- php bin/console doctrine:migration:migrate
c- php bin/console doctrine:migrations:migrate
d- php bin/console make:migration

6- Quelle est la commande pour ajouter un nouvel attribut à une entité nommée **Summer** ?
a- php bin/console create:entity Summer
b- php bin/console update:entity Summer
c- php bin/console make:entity Summer
d- php bin/console add:attribute Summer

B. Question/Réponse (4 pts)

1- Convertir le code PHP suivant en Twig.

```
<table>
< ?php
{ foreach ($mesRelations as $relation )
   { if ($relation->getDuree > 3 ){
     echo '<tr><td>'. $relation->getNom().'</td>' ;
     echo '<td><img src='. $relation->getPhoto().'/></td></tr>' ;
    }}
?>
</table>
```

……
……
……
……
……
……
……

2- Soient les deux routes successives suivantes :

```
#[Route('/myEx/{id}', name: 'exById')]
public function exById($id){….}

#[Route('/myEx/{name}', name: 'exByName')]
public function exByName($name){….}
```

2.1- Pour chaque URL, préciser quelle méthode sera appelée :

a- '/myEx/3':

……

b- '/myEx/BradleyCooper':

……

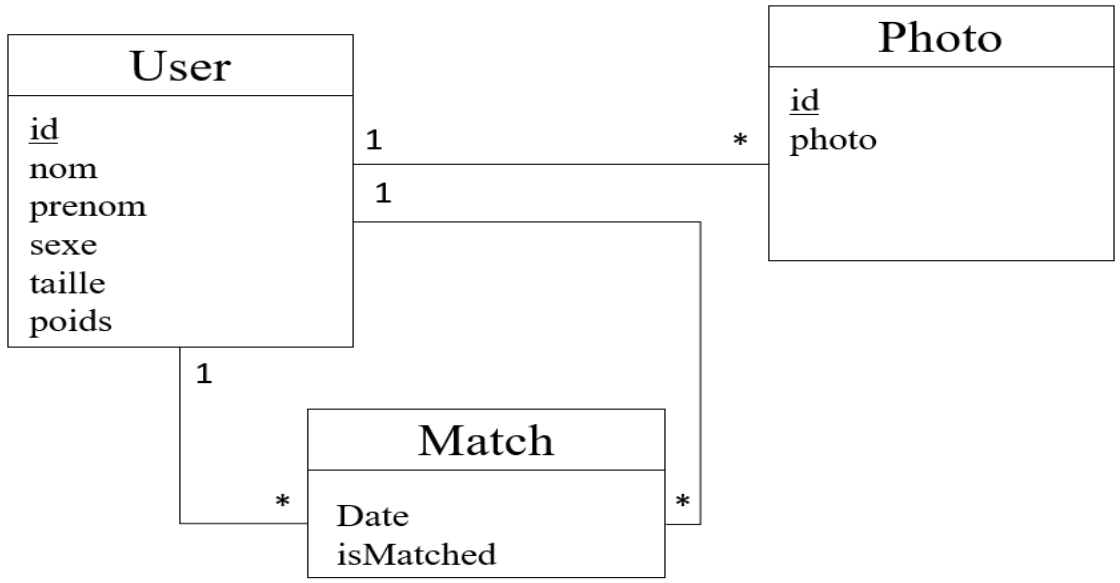
2.2- Réécrire la première route pour quelle intercepte que les id de type entier.

……
……

Problème : (10 pts)

Notre objectif est d'implémenter un site web de rencontre en ligne ***Tek-inder***. Dans ce cadre, on propose de créer une application avec Symfony 6.4.

Soit le diagramme de classes suivant :



A. Configuration des entités :

A.1. Compléter les annotations (et les attributs) manquantes dans les entités suivantes sachant qu'on peut naviguer entre les entités dans les deux sens :

L'attribut « isMatched » est un booléen.

```
<?php
namespace App\Entity;
use Doctrine\ORM\Mapping as ORM;
……
class User
{

 #[ORM\GeneratedValue]
 #[ORM\……]
 #[ORM\……]

   private $id;

#[ORM\……]
   private $nom;

#[ORM\……]
```

```
<?php
namespace App\Entity;
use Doctrine\ORM\Mapping as ORM;
……
class Match
{

 #[ORM\GeneratedValue]
 #[ORM\……]
 #[ORM\……]
   private $id;

#[ORM\……]
  private $date;


  #[ORM\……]
```

```
   private $prenom;

#[ORM\.....]
   private $sexe;

#[ORM\.....]
   private $taille;

#[ORM\.....]
   private $poids;

.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....

//Getters-Setters
}
```

```
   private $isMatched;


.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....

//Getters-Setters
}
```

B. Affichage de la liste des matchs :

Soit ***MatchController***, le contrôleur responsable des actions de gestion des matchs.

Soit ***MatchRepository***, le repository de l'entité match.

B.1. Terminer l'action ***listM()*** qui permet de retourner la page Twig suivante : ***'listM.html.twig'*** (se trouve sous : ***Templates/M***). Cette page Twig permet de retourner la liste des matchs sauvegardés dans notre base de données.

```
<?php
namespace App\Controller;
use Symfony\Bundle\FrameworkBundle\Controller\Controller;

class MatchController extends AbstractController
{
.....
.....
.....
   public function listM(.....)
   {
      //Liste des matchs
```

```
    ……

     return $this->render('……',
        array(
        ……
        ));
   }
}
```

B.2. Compléter la page Twig suivante : ***'listM.html.twig'***. Afficher les noms des deux utilisateurs d'un match.

```
<h1> Liste des matchs </h1>
<table border="1">
   <tr>
      <td> id </td>
      <td>…… </td>
      <td>…… </td>
      <td>…… </td>
      <td>…… </td>
   </tr>
   ……
   ……
   ……
   ……
   ……
   ……
   ……
   ……
   ……
   ……
   ……
   ……
   ……
   ……
   </table>
```

C. Ajout d'un User :

Pour ajouter un utilisateur, nous avons besoin de créer un formulaire pour introduire ses données. Soit ***'UForm'***, le formulaire d'ajout d'un utilisateur.

Soit ***UController***, le contrôleur responsable des actions de gestion des utilisateurs.

C.1. Terminer l'action ***addU()*** qui permet de:

- Rediriger vers la page d'authentification ***Si*** l'ajout d'un user est effectué correctement (sachant que le nom de la route de la page d'authentification est ***signinRoute***).
- ***Sinon***, Afficher le formulaire ***'UForm'*** en utilisant la page Twig suivante : ***'addU.html.twig'*** (se trouve sous : ***Templates/U***).

- ***Créer et affecter la photo « avatar.png » à l'utilisateur à ajouter.***

```
<?php
namespace App\Controller;
//use
class UController extends AbstractController
{
……
……
……
 public function addU(Request ……, ……)
   {

     // Création d'un user
      ……

     // Création d'un formulaire
      $form = $this->…… (……::class, ……);
      $form->handleRequest(……);

     if (……)
      {
        //Ajout de l'utilisateur
         ……
         ……
         ……
         ……
         ……
         ……
         return $this-> ……
      }

      return $this->render('……',
         array(
            ……
         ));
   }
}
```

C.2. Compléter la page Twig suivante : ***'addU.html.twig'***.

```
<h1>Add User</h1>

<form action=…… method=……..>

   ……

   <input type=…… />
</form>
```

D. Re-Conception et Ajout d'un User :

D.1. Re-Modéliser cette application en prenant en considération les besoins suivants :

- L'utilisateur peut avoir plusieurs intérêts.

- Un intérêt peut être partagé par plusieurs utilisateurs.

- Un intérêt est caractérisé par un id et un nom. (exemple d'intérêts : music, sport, mediation,…)

……
……
……
……
……
……
……
……
……
……

D.2. Compléter le formulaire ***'User*Form'*** :

Soit ***'User*Form'***, le formulaire d'ajout d'un user.

Afficher les **noms des intérêts** pour **la liste déroulante des intérêts**.

```
<?php
namespace App\Form;

class UserForm extends AbstractType
{
   public function buildForm(FormBuilderInterface $builder, array $options)
   {
      $builder
         ->add(……., …… ::class)
……
……
……
……
……
   }
}
```

E. Doctrine Query Language (DQL) :

E.1. Ecrire une action php **mesMatchs($id)** permettant de calculer le nombre des matchs **validés** (acceptés) d'un utilisateur identifié par **$id** et qui partage l'intérêt ***« harry potter »*** avec ses matchs groupés par **sexe**.

Exemple :

L'utilisateur « John » (identifié par id=2) a comme matchs :

Male, 25

Female, 3

```
Public function mesMatchs ($id,  …… ) {

……
……
……
……
……
……
……
……
……
……
……
……
……
……
……
……
……
……
……
……
……
……
……

}
```

Annexe :

Pour afficher une liste déroulante de type Entity dans un formulaire, utiliser le code suivant :

EntityType::class, array('class' => 'App\Entity\NomEntity', 'choice_label'=>'NomLabel', 'expanded'=>false, 'multiple'=>false)