CONTRÔLE DES CONNAISSANCES N° 2 Vendredi 01 octobre 2021
DURÉE : 1h

NOM/Prénom : /40 /20

PARTIE 1 - MVC Web

Q1. - voici un extrait de code (un contrôleur) de la classe `UserController :`

```
 /**
 * @Route("/users", name="_user_list")
 */
public function xxxAction(){
  $users = array();
  $obj = new Person("prenom1","nom1");
  $users[] = $obj;
  $obj = new Person("prenom2","nom2");
  $users[] = $obj;
  $obj = new Person("prenom3","nom3");
  $users[] = $obj;
   return this->render('user/xxx.html.twig', array(
        'drupalUsers' => $users));
}
```

Q1.1 Combien d'éléments contient le tableau $users ? (2 points)

Le tableau $Users contient 3 éléments

Q1.2 Où le code de la vue est-il situé dans l'arborescence du projet ? (2 points)

Arborescence correspond a la structure d'un dossier sur un disque.
L'arborescence est src/templates/user

Q1.3 Donner l'instruction côté vue qui donne le nombre d'utilisateurs reçu. (voir doc ci-dessous) (2 points)

{{drupaluser | length}}
ou
{ % set nbUser = drupalUSer | length%}
{{nbUser}}

Exemple de documentation technique twig

Twig template: find the length of an array

It's not immediately apparent in the documentation for Twig, but finding the size of an array in a twig template is simple:

```
{% if my_list|length > 10 %}
    ...
{% endif %}
```

Or, you can extract the size into a variable:

```
{% set size = my_list | length %}
```

Q2. - Un développeur de votre équipe découvre Symfony. Il souhaite solliciter l'url suivante :

`http://localhost:8000/demo/bonjour`

Mais reçoit le message suivant :

Q2.a Expliquez la raison **précise (ne pas répéter la cause de l'erreur)** de cette erreur. **(2 points)**

La méthode *controller* associée à une route n'est pas définie ou n'existe pas.

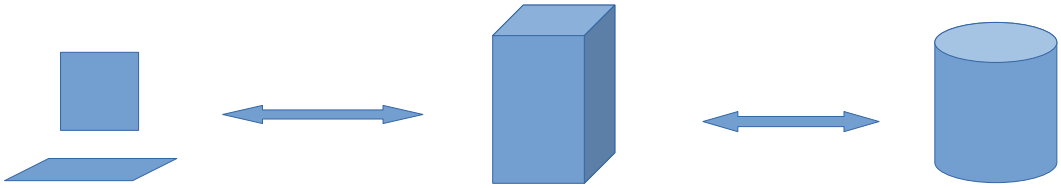
Q2.b Concevoir le code d'un contrôleur (le plus simple possible) qui éviterait cette erreur**.**
(3 points)

```
/**
* @route (”demo /bonjour", name = ”bonjour”)
*/

Public fonction bonjourAction () {
        return new Response("bonjour");  // voir Q4
}
```

Q3. Question d'architecture - (3 points)

Q3.1 A partir de cette architecture 3 tiers Web, placer sous chacun des tiers les termes techniques proposés dans la liste ci-dessous, si vous considérez le ou les concepts, derrière ces termes, sont actifs (comprendre exécutés) sur le tiers en question.



Client Web Navigateur **serveur HTTP** **SGBDR**

JS HTML CSS HTTP PHP, HTTP,Symphony,twig,(JS) SQL

Liste de termes : JavaScript, HTML, HTTP, SQL, PHP, CSS, Twig, Symfony
rem : une techno peut être active sur plus d'un tiers

Q4 Dans cet extrait de code

```php
// src/AppBundle/Controller/LuckyController.php
namespace AppBundle\Controller;

use Sensio\Bundle\FrameworkExtraBundle\Configuration\Route;
use Symfony\Component\HttpFoundation\Response;

class LuckyController
{
    /**
     * @Route("/lucky/number")
     */
    public function numberAction()
    {
        $number = mt_rand(0, 100);

        return new Response(
            '<html><body>Lucky number: '.$number.'</body></html>'
        );
    }
}
```

[x] @Route("/lucky/number") est placé dans un commentaire commençant obligatoirement par `/**` **C'est du php doc**

[x] @Route("/lucky/number") est une façon de lier une *route* à un *contrôleur*

[x] *namespace* permet regrouper logiquement certaines classes entre elles

[] *namespace* est une sorte d'import

[] le commentaire à la ligne 1 est obligatoire

Q5 Dans cet extrait :

```
/**
* @Route("/blog/{slug}", name="blog_show")
*/
public function showAction($slug) { … }
```

[x] le paramètre *$slug* de la méthode doit avoir le même nom que la partie variable de la route
Obligatoire d'avoir le même nom
[ ] le paramètre *$slug* de la méthode n'a rien a voir avec les données de l'URL de la route
[x] le paramètre *$slug* de la méthode sera automatiquement renseigné par le framework

---

Q6 Soit l'extrait suivant :

```
/**
 * @Route("/blog/{page}", name="blog_list", requirements={"page": "\d+"}) Valeur numérique décimal Base 10.
 */
public function listAction($page = "3")
{
    // ...
}
```

[x] le paramètre *$page* a comme valeur par défaut : "3"
[ ] le paramètre *$page* a comme valeur par défaut : "page"
[x] la route */blog/0* provoquera l'exécution de la méthode *listAction* avec 0 en argument
si on lui passe un valeur correcte, la valeur par par défaut saute (plus prise en compte)
[x] la route */blog/3* provoquera l'exécution de la méthode *listAction* avec 3 en argument
[] la route */blog/un* provoquera l'exécution de la méthode *listAction* avec '*un*' en argument

---

Q7 Soit l'instruction :

```
return $this->render('lottery/lucky/number.html.twig',
                      array('name' => $name)
                     );
```

Dans une méthode contrôleur, une telle instruction :

[x] signe la fin de l'exécution du corps de la méthode return signifie que la fonction retourne à l'appelant
[x] prend la responsabilité de désigner la vue à appliquer suite à une requête HTTP via render
[ ] renverra toujours à la même vue, avec la même valeur d'argument
[x] renverra toujours à la même vue, avec une valeur variable d'argument ($name est variable)

Q8 Avec Twig je peux logiquement

[x] générer du HTML
[ ] écrire du code PHP
[x] générer du code CSS
[x] écrire des boucles
[x] écrire du contenu sous condition alternative

[x] afficher la valeur de la variable xx avec l'instruction `{{ xx }}`
[ ] afficher la valeur de la variable xx avec l'instruction `{% xx %}`
[ ] afficher la valeur de la variable xx avec l'instruction `{# xx #}`

Q9 Dans ce code :

```
{% extends 'base.html.twig' %}

{% block title %}My cool blog posts{% endblock %}

{% block body %}
    {% for entry in blog_entries %} entry est une variable locale de boucle
        <h2>{{ entry.title }}</h2>
        <p>{{ entry.body }}</p>
    {% endfor %}
{% endblock %}
```

[x] le bloc *body* est entièrement redéfini *(override) : redéfinition complète*
[ ] le bloc *title* est étendu *(override) : étendre = héritage + ajout*
[x] la variable *blog_entries* a été transmise par le contrôleur
[ ] la variable *entry* a été transmise par le contrôleur
[x] la variable *blog_entries* est une collection
[ ] la variable *entry* est une collection
[x] la variable *entry* référence un objet // notation pointée

Q10 Soit l'extrait suivant

Ce que

```php
// src/AppBundle/Controller/LuckyController.php
namespace AppBundle\Controller;

use Symfony\Component\HttpFoundation\Response;
use Sensio\Bundle\FrameworkExtraBundle\Configuration\Route;

class LuckyController
{
    /**
     * @Route("/lucky/number/{max}")
     */
    public function numberAction($max)
    {
        $number = mt_rand(0, $max);

        return new Response(
            '<html><body>Lucky number: '.$number.'</body></html>'
        );
    }
}
```

retourne la méthode d'action *numberAction* est :

[ ] toujours le même nombre
[x] une référence à un objet
[x] une référence à une instance de `HttpFoundation\Response`
[ ] une référence à une chaîne de caractères

Q11 Soit une méthode d'une classe XxxController, qui hérite de la classe AbstractController, contenant l'instruction `$this->redirectToRoute(...)`

[ ] `redirectToRoute` peut être appelée sans être préfixée par `$this->`
[x] `$this` fait référence à une instance en cours de XxxController
[x] l'expression (`$this instanceof XxxController`) rend **true**
[ ] l'expression (`$this instanceof AbstractController`) rend **false**

Q12 Un message *flash* est

[ ] un message de notification à destination du serveur
[x] un message de notification à destination du client
[x] un message à usage unique initié côté serveur
[ ] un message à usage unique initié côté client
[x] un message au cycle de vie court (le temps d'un affichage dans la session utilisateur)
[ ] un message au cycle de vie moyen (le temps d'une session utilisateur)

Q13 Une instance de *Request* (Symfony\Component\HttpFoundation\Request), me permet

[x] de récupérer les données d'entrée en GET
[x] de récupérer les données d'entrée en POST
[x] d'accéder aux données de la session (s'il y en a) message flash par exemple
[x] d'ajouter des données à la session utilisateur
[x] d'obtenir des informations sur le client à l'origine de la requête HTTP
[x] de lire et de créer des cookies utilisateur

Q14 Du point de vue du client, une « page web » est accessible :

[x] via une *url*
[ ] via un contrôleur principal
[ ] via une vue *twig*
[ ] via une méthode d'une classe contrôleur

Q15 Dans un fichier Twig, une telle commande `{{ title }}` est l'équivalent :

[x] d'un `echo` en php
[ ] d'un `var_dump` en php
[ ] du placement d'une balise `title` HTML
[x] de l'insertion, en lieu et place de l'instruction, de la valeur détenue par la variable `title`

Q16 Soit l'instruction :

```
return $this->render('lottery/lucky/number.html.twig',
                     array('name' => $name)
                    );
```

Une telle instruction se trouve généralement :

[ ] dans un constructeur
[x] dans un contrôleur
[x] dans une méthode d'une classe contrôleur
[ ] dans une vue

Q17 Quel est l'équivalent de la variable super-globale $_GET en symfony, lorsque, par exemple on souhaite obtenir une donnée ayant pour clé 'nom' de cette façon : $nom = $_GET['nom'] ; Sachant que $request est instance de la classe Request (de HttpFoundation)

```
[ ] y a pas
[ ] $nom = Request->get('nom');
[ ] $nom = query->get('nom');
[ ] $nom = $request->request->get('nom');
[x] $nom = $request->query->get('nom');
[ ] $nom = $this->get('nom');
```

PARTIE 3 – Test unitaire (8 points)

Voici le corps d'une méthode de test : l'objet à tester est référencé par la variable $obj :

```
String $s = $obj->m();
assertEquals("sio", $s);
```

Q18 Quel est l'objectif de ce test ? (**4 points**)

l'objectif de ce test est de savoir si l'appelle de la méthode de *m* sans argument de l'objet référencé par l'objet $obj renvoie bien la chaîne "sio"

Q19 En vous aidant de l'extrait de l'API xUnit donné, réécrire ce code en utilisant la méthode `assertTrue` au lieu de `asserEquals`. (**4 points**)

API (extrait) de PHPUnit

```
assertEquals($expected, $actual)
  // Asserts that two objects are equal.

assertTrue(boolean $exprbool)
  // Asserts that $exprbool is true
```

```
assertTrue("sio" === $s) // passe une valeur booléenne à assertTrue
```

PARTIE 4 – ORM (4 points)

Voici le code d'une entité

```
/**
 * @ORM\Entity(repositoryClass=ProductRepository::class)
 */
class Product
{
    /**
     * @ORM\Id
     * @ORM\GeneratedValue
     * @ORM\Column(type="integer")
     */
    private $id;

    /**
     * @ORM\Column(type="string", length=255)
     */
    private $name;
}
```

Q20 L'attribut $id :

[ ] aura comme valeur une valeur de clé primaire
[x] aura comme valeur une valeur générée par le SGBDR
[x] aura comme valeur une valeur générée par Doctrine
[x] aura comme valeur une valeur générée par l'ORM
[ ] aura comme valeur une valeur générée par l'instance
[ ] aura comme valeur une valeur générée par le constructeur

Q21 L'attribut $name :

[x] aura comme valeur une valeur gérée par le SGBDR
[x] aura comme valeur une valeur gérée par le Doctrine
[x] aura comme valeur une valeur gérée par l'ORM
[x] aura comme valeur une valeur gérée par l'instance Important l'élément principal
[x] aura comme valeur une valeur gérée par le constructeur

Q22 La table correspondante

[ ] sera composée de 3 colonnes
[x] sera composée de 2 colonnes $id, nom
[x] aura une clé primaire id
[ ] aura une clé étrangère
[ ] aura au moins une ligne

Q23 La table correspondante

[x] pourra être mise à jour si la classe Product évolue
[ ] ne pourra pas être mise à jour si la classe Product évolue
[ ] aura toujours comme nom de colonnes les noms des attributs
[x] ne pourra pas toujours avoir comme nom de colonnes les noms des attributs