المعهد العالي للدراسات التكنولوجية بمدنين
Institut Supérieur des Etudes Technologiques de Médenine

Atelier Framework Coté Serveur

TP2

Eléments de base
(Symfony 6)

Enseignante : K.MECHLOUCH	Classe : L2DSI

1. Préparation de l'environnement du travail

1.1. Les environnements

Lorsque vous développez une application, sauf si vous développez directement sur le serveur de production, vous avez besoin de plusieurs environnements :

- L'**environnement de développement** : c'est l'environnement utilisé par les **développeurs Web** quand ils travaillent sur l'application pour ajouter de nouvelles fonctionnalités, corriger des bugs, ...
- L'**environnement de test** : cet environnement est utilisé pour tester automatiquement l'application.
- L'**environnement de qualité** : cet environnement est utilisé par le **client** pour tester l'application et les bogues ou les fonctionnalités manquantes.
- L'**environnement de production** : c'est l'environnement où interagissent les **utilisateurs finaux.**

1.2. Recettes d'installation automatique avec Symfony Flex

Lors de l'exécution de *composer require annotations,* deux choses particulières se produisent grâce à un puissant plugin Composer appelé *Flex* :

- Après le téléchargement de ce package, *Flex* exécute une recette, qui est un ensemble d'instructions automatisées indiquant à Symfony comment intégrer un package externe. Les recettes *Flex* existent pour de nombreux packages et ont la capacité de faire plusieurs tâches, comme ajouter des fichiers de configuration, créer des répertoires, mettre à jour *.gitignore* et ajouter une nouvelle configuration à votre fichier .env.
  =>*Flex* automatise l'installation des packages.

1.3. Installation des packages

Une pratique courante lors du développement d'une application Symfony consiste à installer des packages (Symfony les appelle des bundles) qui fournissent des fonctionnalités prêtes à l'emploi.

Exemple :

```
cd my-project/
composer require logger
```

Parfois, une seule fonctionnalité nécessite l'installation de plusieurs packages et bundles. Au lieu de les installer individuellement, Symfony fournit des packs, qui sont des métapaquets Composer qui incluent plusieurs dépendances.

***Exemple* :**

Pour ajouter des fonctionnalités de débogage dans votre application, vous pouvez exécuter la commande :

```
composer require --dev debug
```

Cela installe le *symfony/debug-pack*, qui à son tour installe plusieurs packages comme *symfony/debug-bundle*, *symfony/monolog-bundle*, *symfony/var-dumper*, etc.

1.4. La commande bin/console

Il y a une liste de commandes qui peuvent vous donner des informations de débogage, aider à générer du code, générer des migrations de base de données et bien plus encore.

Exemple : pour obtenir une liste de toutes les routes de votre système, utilisez la commande *debug:router* :

```
php bin/console debug:router
```

On peut lancer les commandes via l'invocation **symfony console** de l'outil Symfony en ligne de commande préalablement installé.

1. Pour vérifier la version de Symfony utilisée dans le projet

```
symfony console -V
```

```
Symfony 6.3.3 (env: dev, debug: true) [...]
```

Vous êtes en environnement de développement dev (pas encore prêts pour la production !)

2. Pour afficher la liste des sous-commandes disponibles (avec la sous-commande list, ou directement sans argument passé à symfony console) :

```
symfony console list
```

3. Pour consulter l’aide en ligne avec la sous-commande help :

```
symfony console help about
```

4. Pour avoir des informations sur l’environnement de développement et de mise au point :

```
symfony console about
```

1.5. La barre d'outils de débogage Web : Debugging Dream

L'une des fonctionnalités de Symfony est la barre d'outils de débogage Web : une barre qui affiche une énorme quantité d'informations de débogage en bas de votre page pendant le développement. Tout cela est inclus dans un pack appelé ***symfony/profiler-pack***.

1.6. Vérification des vulnérabilités de sécurité

La commande utilisée pour vérifier si les dépendances de votre projet contiennent une vulnérabilité de sécurité connue :

```
symfony check:security
```

1.7. Configuration d’un serveur web

Pour installer le pack apache exécutez la commande suivante :

```
composer require symfony/apache-pack
```

Ce pack crée un fichier *.htaccess* dans le répertoire *public/* qui contient les règles nécessaires pour servir l'application Symfony.

2. Les bundles

Les fonctionnalités de base du framework Symfony sont implémentées avec des bundles (FrameworkBundle, SecurityBundle, DebugBundle, etc.) Ils sont également utilisés pour ajouter de nouvelles fonctionnalités dans votre application.

Un bundle peut envelopper toutes les fonctionnalités trouvées dans une application Symfony, y compris la configuration, les contrôleurs, les routes, les services, les écouteurs d'événements, les modèles, etc.

Dans les versions de Symfony antérieures à 4.0, il était recommandé d'organiser l'application à l'aide de bundles. Ce n'est plus recommandé et les bundles ne doivent être utilisés que pour partager du code et des fonctionnalités entre plusieurs applications.

Vous allez créer et activer un nouveau bundle IsetTestBundle :

Commencez par créer un répertoire src/Iset/TestBundle/ et ajoutez un nouveau fichier appelé IsetTestBundle.php :

```
// src/Iset/TestBundle/IsetTestBundle.php
namespace App\Iset\TestBundle;
use Symfony\Component\HttpKernel\Bundle\Bundle;
class IsetTestBundle extends Bundle
{
}
```

Cette classe vide est la seule pièce dont vous avez besoin pour créer le nouveau bundle. Bien que généralement vide, cette classe est puissante et peut être utilisée pour personnaliser le comportement du bundle. Maintenant que vous avez créé le bundle, activez-le :

```
// config/bundles.php
return [
App\Iset\TestBundle\IsetTestBundle::class => ['all' => true],
];
```

3. Les pages

La création d'une nouvelle page s'effectue en deux étapes :

- **Création d'une route** : une route est l'URL (par exemple /home) de votre page et pointe vers un contrôleur ;
- **Création d'un contrôleur** : un contrôleur est la fonction PHP que vous écrivez et qui construit la page. Le contrôleur reçoit la requête demandée (Request) et crée un objet Symfony (Response) qui peut contenir du contenu HTML, une chaîne JSON ou même un fichier binaire comme une image ou un PDF.

3.1.Contrôleur

Créez le contrôleur suivant dans src/Controller/HelloController.php :

```
<?php
 namespace App\Controller;
use Symfony\Component\HttpFoundation\Response;
class HelloController
```

```
{
  public function sayHello()
  {
    return new Response('Hello!');
  }
}
```

Vous pouvez utiliser la commande :

```
composer require --dev symfony/maker-bundle
php bin/console make:controller HelloController
```

3.2.Les routes

Il y a plusieurs méthodes pour définir les routes :

- En utilisant le fichier routes.yaml.
- En utilisant les annotations.
- En utilisant : #[Route('/path', name: 'path_name')]

a. Fichier routes.yaml

Vous devez maintenant associer la fonction *sayHello* à une URL publique (par exemple, /salutation) afin que la méthode *sayHello* () soit appelée lorsqu'un utilisateur y accède. Cette association est définie en créant une route dans le fichier ***config/routes.yaml*** :

```
#nom de la route
Hello_page:
  path: /salutation
  controller: App\Controller\HelloController::sayHello
```

Test: http://localhost:8000/salutation

b. Annotations

Au lieu de placer toutes les routes de l'application dans un seul fichier, il peut-être plus souple d'utiliser les *annotations* dans le code même du contrôleur pour plus de commodité :

- Commencez par les installer :

```
composer require annotations
```

- Puis annotez votre contrôleur :

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

```php
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Routing\Annotation\Route;

class HelloController
{
  /**
   * @Route("/salutation", name="hello_page")
   */
  public function sayHello()
  {
    return new Response('Hello!');
  }
}
```

c. Routes paramétrées

Ajoutez une autre route paramétrée dans votre contrôleur :

```php
class HelloController
{
  /**
   * @Route("/salutation", name="hello_page")
   */
  public function sayHello()
  {
    return new Response('Hello!');
  }
  /**
   * @Route("/bonjour/{nom}")
   */
  public function bonjour($nom)
  {
    return new Response("Bonjour $nom !");
  }
}
```

d. Debug des routes

On peut lister toutes les routes avec la commande suivante :

```
php bin/console debug:router
```

Exercice

1. Créez une page - /nombre/aleatoire - qui génère un numéro aléatoire et l'imprime.
2. Créez une page qui affiche : Bonjour Mr/Mme *nom, votre numéro* est *numéro.*