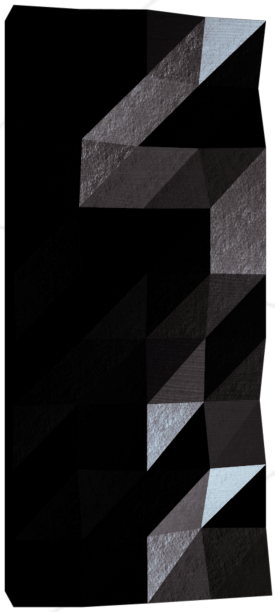

Formation PHP Symfony - 3

Symfony Avancés

Résumé: Cette journée est consacrée à l'apprentissage de concepts avancés de Symfony tels que les traductions, les configurations personnalisées de bundles et les tests unitaires.

Version: 1

Table des matières

Chapitre I

Préambule

Jusqu'à présent, vous avez été initiés à Symfony et son fonctionnement de base, SQL et ORM, vous avez appris à propos de l'authentification mais beaucoup reste à découvrir. Aujourd'hui nous allons découvrir de nouveaux concepts avancés utilisés au quotidien par les développeurs Symfony.

Chapitre II

Règles communes

- Votre projet doit être réalisé dans une machine virtuelle.
- Votre machine virtuelle doit avoir tout les logiciels necessaire pour réaliser votre projet. Ces logiciels doivent être configurés et installés.
- Vous êtes libre sur le choix du systmème d'exploitation à utiliser pour votre machine virtuelle.
- Vous devez pouvoir utiliser votre machine virtuelle depuis un ordinateur en cluster.
- Vous devez utiliser un dossier partagé entre votre machine virtuelle et votre machine hote.
- Lors de vos évaluations vous allez utiliser ce dossier partager avec votre dépot de rendu.
- Vos fonctions de doivent pas s'arrêter de manière inattendue (segmentation fault, bus error, double free, etc) mis à part dans le cas d'un comportement indéfini. Si cela arrive, votre projet sera considéré non fonctionnel et vous aurez 0 au projet.
- Nous vous recommandons de créer des programmes de test pour votre projet, bien que ce travail **ne sera pas rendu ni noté**. Cela vous donnera une chance de tester facilement votre travail ainsi que celui de vos pairs.
- Vous devez rendre votre travail sur le git qui vous est assigné. Seul le travail déposé sur git sera évalué. Si Deepthought doit corriger votre travail, cela sera fait à la fin des peer-evaluations. Si une erreur se produit pendant l'évaluation Deepthought, celle-ci s'arrête.

Chapitre III

Règles spécifiques de la journée

- Pour cette journée, votre repository doit contenir seulement une application Symfony.
- Vous devrez respecter les bonnes pratiques du framework Symfony.
- Si aucune information contraire n'est explicitement présente, vous devez assumer les versions de langages suivantes :
  - `PHP - Symfony version LTS`
  - `HTML 5`
  - `CSS 3`

Chapitre IV

Exercice 00

	Exercice :
Exercice 00 : Bundle personnalisé basique	
Dossier de rendu : *ex/*	
Fichiers à rendre : `Fichiers et répertoires de votre application`	
Fonctions Autorisées : `Toutes méthodes`	

Pour cet exercice vous devez mettre en place une application Symfony et un bundle personnalisé qui sera étendu au fur et à mesure par les exercices suivants. Le bundle s'appellera **D07Bundle** et c'est le seul bundle qui se trouvera dans votre application. Le répertoire du bundle ne devra pas contenir d'autres répertoires à part un répertoire vide pour le contrôleur pour le moment. Aussi, le répertoire **src/D07Bundle/Resources/views** devra être vide pour le moment.
Vous devrez obtenir une erreur 404 si vous essayez d'accéder à votre site sur le chemin par défaut.

Chapitre V

Exercice 01

	Exercice :
Exercice 01 : Configuration du Bundle	
Dossier de rendu : *ex/*	
Fichiers à rendre : `Fichiers et répertoires de votre application`	
Fonctions Autorisées : `Toutes méthodes`	

Pour cet exercice vous ajouterez des fonctionnalités avancées à votre bundle. Vous êtes probablement déjà familiers avec les configurations système de Symfony et ce qu'est un bundle. Chaque bundle peut avoir sa propre configuration. Premièrement vous devez créer un fichier de configuration pour votre bundle. La configuration de votre bundle devrait avoir la clé racine (root key) **d07** et les 2 sous-clés suivantes :

- **number** - obligatoire, devrait être un entier
- **enable** - facultatif, devrait être un boolean, ayant par défaut la valeur true

Apres avoir mis en place le fichier de configuration pour le bundle, ajouter les clés obligatoires de votre configuration bundle à votre fichier **config/packages/bundle.yml**.

Afin de tester que votre configuration fonctionne correctement, créer un nouvelle classe contrôleur nommée **Ex01Controller** avec une action nommée **ex01Action** avec la route **/ex01**. Cette route devra simplement retourner un texte plein contenant la valeur de la sous-clé **number** de votre configuration du bundle.

Astuce : *Le contrôleur dispose d'une fonction* ***getParameter*** *afin de retourner n'importe quel paramètre du container. Cependant, assurez-vous que la configuration de votre bundle est disponible dans le container. Peut-être qu'une classe Extension peut aider ?*

Chapitre VI

Exercice 02

	Exercice :
Exercice 02 : Traductions	
Dossier de rendu : *ex/*	
Fichiers à rendre : `Fichiers et répertoires de votre application`	
Fonctions Autorisées : `Toutes méthodes`	

Il est temps de rendre votre application capable d'afficher des contenus dans plusieurs langues, **en** et **fr**. Paramétrer le *locale* par défaut et activer les traductions. Créer les nouveaux fichiers dans les répertoires de votre application. Ils s'appelleront **messages.en.yml** et **messages.fr.yml**.

Créer un contrôleur personnalisé **Ex02Controller** avec une action personnalisée **translationsAction** qui prendra un paramètre facultatif **count**, entier, avec une valeur possible dans l'intervalle de *0 à 9*, et ayant 0 comme valeur par défaut. La route pour cette action sera **/{_locale}/ex02/{count}**. Selon la valeur du paramètre **_locale**, le site sera affiché soit en anglais soit en français. Maintenant créer un template appelé **ex02.html.twig** que sera retourné par votre action. Le template aura comme paramètres le nombre trouvé dans la configuration de votre bundle décrit dans l'exercice précédent ainsi que le paramètre **count**.

Le template devra contenir le texte suivant selon la langue et le texte devrait être récupéré à partir des fichiers de traduction .

Anglais

- *The config number is %number%* - le paramètre correct devra être passé à cette traduction
- *none/one/number %count%* - selon la valeur du paramètre une traduction différente sera utilisée

Français

- *Le numéro de configuration est %number%* - le paramètre correct devra être passé à cette traduction

- *aucun/un/nombre %count%* - selon la valeur du paramètre une traduction différente sera utilisée

Astuce : *Utilisez les filtres Twig pour traductions et l'annotation* ***@Route*** *pour la validation des paramètres.*

Chapitre VII

Exercice 03

	Exercice :
Exercice 03 : Extension Twig & Injection de dépendance	
Dossier de rendu : *ex/*	
Fichiers à rendre : `Fichiers et répertoires de votre application`	
Fonctions Autorisées : `Toutes méthodes`	

Je parie que vous vous demandez comment Twig fonctionne et comment ajouter vos propres fonctions et filtres à utiliser dans vos templates. Pour cela vous pouvez créer une extension Twig. La classe pour cette extension devra être **src/D07Bundle/Twig/Ex03Extension.ph**
Créer un nouveau filtre Twig et une fonction Twig. Le filtre s'appellera **uppercaseWords** et lorsqu'il est appliqué à une chaine de caractères il devra transformer en majuscule la première lettre de chaque mot. La fonction s'appellera **countNumbers** et retournera le nombre de chiffres dans une chaine de caractères.
Ces fonctions ne devraient pas être créées dans la classe extension mais dans une classe séparée de service **src/D07Bundle/Service/Ex03Service**. Cette classe de service sera ensuite injectée dans l'extension Twig.
Ensuite créer un contrôleur appelé **Ex03Controller** ayant une action **extensionAction**, la route **/ex03** et un template **ex03.html.twig**. Le template devra utiliser le filtre et la fonction sur la chaine de caractères que vous voulez en provenance des traductions et afficher les résultats.

Astuce : *Les services peuvent être définis dans le fichier* ***config/services.yml****.*

Chapitre VIII

Exercice 04

	Exercice :
Exercice 04 : Tests unitaires	
Dossier de rendu : *ex/*	
Fichiers à rendre : `Fichiers et répertoires de votre application`	
Fonctions Autorisées : `Toutes méthodes`	

Il est temps de tester le service créé dans l'exercice précédent à l'aide de **PHPUnit** et des tests unitaires. Créer une classe pour votre service qui devra suivre la convention de nommage de tests Symfony et ensuite tester les 2 fonctions , **uppercaseWords** et **countNumbers**, dans ce service.
Ecrire au moins 3 tests différents pour chaque fonction et assurez-vous de respecter les conseils de l'exercice précédent.
Tous les tests doivent passer !

Chapitre IX

Rendu et peer-évaluation

Rendez votre travail dans votre dépôt `Git` comme d'habitude. Seul le travail présent dans votre dépôt sera évalué en soutenance. Vérifiez bien les noms de vos dossiers et de vos fichiers afin que ces derniers soient conformes aux demandes du sujet.

`L'évaluation se déroulera sur l'ordinateur du groupe évalué.`