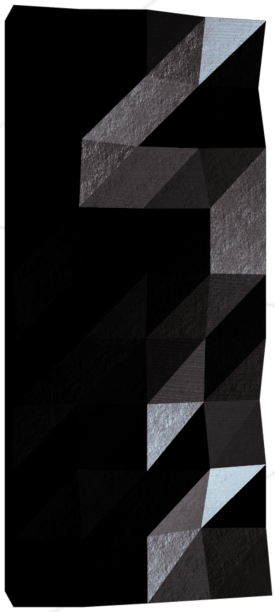

Formation PHP - Symfony

D04 - Getting started with Symfony

Alexandru Puscas alexandru.puscas@pitechplus.com
42 Staff pedago@staff.42.fr

Résumé: Following this day you will get to know basic concepts in Symfony framework.

Table des matières

Chapitre I

Préambule

The void type, in several programming languages derived from C and Algol68, is the type for the result of a function that returns normally, but does not provide a result value to its caller. Usually such functions are called for their side effects, such as performing some task or writing to their output parameters. The usage of the void type in such context is comparable to procedures in Pascal and syntactic constructs which define subroutines in Visual Basic. It is also similar to the unit type used in functional programming languages and type theory.

Chapitre II

Consignes

Sauf contradiction explicite, les consignes suivantes seront valables pour tous les jours de cette Piscine.

- Seul ce sujet sert de référence : ne vous fiez pas aux bruits de couloir.
- Le sujet peut changer jusqu'à une heure avant le rendu.
- Les exercices sont très précisément ordonnés du plus simple au plus complexe. En aucun cas nous ne porterons attention ni ne prendrons en compte un exercice complexe si un exercice plus simple n'est pas parfaitement réussi.
- Attention aux droits de vos fichiers et de vos répertoires.
- Vous devez suivre la procédure de rendu pour tous vos exercices. L'url de votre dépot `GIT` pour cette journée est disponible sur votre intranet.
- Vos exercices seront évalués par vos camarades de Piscine.
- En plus de vos camarades, vous pouvez être évalués par un programme appelé la Moulinette. La Moulinette est très stricte dans sa notation car elle est totalement automatisée. Il est donc impossible de discuter de sa note avec elle. Soyez d'une rigueur irréprochable pour éviter les mauvaises surprises.
- Les exercices shell doivent s'éxcuter avec `/bin/sh`.
- Vous ne devez laisser aucun autre fichier que ceux explicitement specifiés par les énoncés des exercices dans votre dépot de rendu.
- Vous avez une question ? Demandez à votre voisin de droite. Sinon, essayez avec votre voisin de gauche.
- Toutes les réponses à vos questions techniques se trouvent dans les `man` ou sur Internet.
- Pensez à discuter sur le forum Piscine de votre Intra et sur Slack !
- Lisez attentivement les exemples car ils peuvent vous permettre d'identifier un travail à réaliser qui n'est pas précisé dans le sujet à première vue.
- Réfléchissez. Par pitié, par Thor, par Odin !

Chapitre III

Règles spécifiques de la journée

- Chaque exercice doit être résolu dans un bundle différent.
- Chaque controller doit aller dans le répertoire Controller à l'intérieur du bundle.
- Les noms de fichiers contenant les classes controller devraient contenir le suffixe `Controller` et devraient contenir une classe du même nom.
- Le contenu de chaque page devrait se trouver dans des fichiers twig et chaque fichier twig devrait avoir l'extension `.html.twig`
- Le serveur utilisé aujourd'hui est celui intégré dans Symfony. Il devrait être démarré et arrêté à l'aide des commandes console Symfony.
- Seulement les URLs explicitement demandés devraient afficher une pages sans erreur. Les URLs non configurées devrait générer une erreur 404.
- Les URLs demandés devraient fonctionner avec ou sans le slash final. Ex : `/ex00` et `/ex00/` doivent fonctionner de manière identique

Chapitre IV

Exercice 00

	Exercice : 00
Exercice 00 : Première page.	
Dossier de rendu : $ex00/$	
Fichiers à rendre : `Tous les fichiers de l'application.`	
Fonctions Autorisées : `Toute fonctionnalité de Symfony.`	
Remarques : `n/a`	

Dans cet exercice vous devez créer une simplemage dans une application Symfony.

Premièrement, créez un projet Symfony à l'aide du composer :

```
composer create-project symfony/framework-standard-edition d04
```

Créer un bundle `E00Bundle` et référencez le dans le fichier AppKernel.

`Vous pouvez créer le bundle à l'aide des commandes natives de symfony.`

Pour définir les routes vous devrez utiliser les annotations dans le fichier controleur (controller) et vous devriez référencer les routes dans le fichier `app/config/routing.yml`.

Pour cet exercice vous devez créer une page qui est visible à l'URL suivante : `/e00/firstpage`. Dans la page vous devez afficher la chaine suivante : `"Hello world!"`.

Vous ne devez utiliser aucun template, vous devez avoir seulement le controller et utiliser l'objet `Response` du composant HttpFoundation.

Chapitre V

Exercice 01

	Exercice : 01
Exercice 01 : Pages multiples.	
Dossier de rendu : $ex01/$	
Fichiers à rendre : `Tous les fichiers de l'application.`	
Fonctions Autorisées : `Toute functionnalité de Symfony.`	
Remarques : `n/a`	

A l'aide du projet créé dans l'exercice précédent, créer un nouveau bundle `E01Bundle` et référencez le dans le fichier AppKernel.

`Vous pouvez créer le bundle à l'aide des commandes natives de symfony.`

Pour définir les routes vous devrez utiliser les annotations dans le fichier controleur (controller) et vous devriez référencer les routes dans le fichier `app/config/routing.yml`.

Vous devez créer une application qui affichera des `cheat sheet` à partir du répo `https://github.com/davidpv/symfony2cheatsheet`. Seulement les fichiers de type `.html.twig` seront intégrés dans l'application.

L'application devrait contenir une page principale à l'adresse `/e01` où vous devez lister tous les liens vers les (sous)pages qui seront accessibles à travers une URL structuré comme suit : `/e01/category`. Par exemple : `/e01/cache` devra afficher le contenu de https ://github.com/davidpv/symfony2cheatsheet/blob/master/cache.html.twig

Les catégories qui seront accesibles sont : controller, routing, templating, doctrine, testing, validation, forms, security, cache, translations, services.

Dans le cas où une URL avec une catégorie inexistante est appelée, l'application devrait afficher la page principale. Ainsi, si vous accédez à `/e01/wrongcategory` vous devrez

voir la page principale avec le liste des liens vers les pages des catégories.

Vous pouvez définir la liste des catégories dans un tableau/array ç l'intérieur du controller ou bien utiliser les paramètres de réglage dans le fichier `Resources/config/services.yml` du Bundle.

Afin de permettre à vous pages de produire du HTML correct vous devez créer un fichier `base.html.twig` qui sera votre template de base et qui contiendra les balises `<html>` et `<body>` de la page. Toutes les pages des catégories devraient utiliser ce template en surchargeant le block `content` du template de base.

Chapitre VI

Exercice 02

	Exercice : 02
Exercice 02 : Premier formulaire.	
Dossier de rendu : $ex02/$	
Fichiers à rendre : `Tous les fichiers de l'application.`	
Fonctions Autorisées : `Toute functionnalité de Symfony.`	
Remarques : `n/a`	

En utilisant le projet créé dans le premier exercice, créer un nouveau bundle `E02Bundle` et référencez le dans le fichier AppKernel.

Vous savez maintenant déjà comment définir les routes.

Dans cet exercice vous aurez à l'URL `/e03` un formulaire qui aura deux inputs : une zone de texte libellé "Message" contenant un message et une liste déroulante libellée "Include timestamp" avec deux valeurs sélectionnables : *Yes* et *No*. Les informations envoyés par le formulaire devront être écrites dans un fichier selon la logique suivante : si l'option "Yes" est sélectionnée l'application écrira le message et le timestamp sur une ligne du fichier ; si l'option "No" est sélectionnée seulement le message sera écrit sur une ligne dans le fichier.

Vous devrez ajouter une validation du message côté serveur afin de s'assurer que le message n'est pas vide. Attention, vous ne devrez pas utiliser la validation HTML5 côté client.

Le nom du fichier dans lequel seront stockés les messages sera défini dans `app/config/parameters.yml` et devra être créé dans le répertoire racine du projet aux côtés des fichiers composer.lock et composer.json. Si le fichier n'existe pas l'application ne devra pas planter mais elle devra créer le fichier à l'endroit indiqué avec le nom spécifié dans `app/config/parameters.yml`.

Le formulaire pourra être envoyé plusieurs fois, le contenu du fichier devra alors intégrer les nouveaux éléments envoyés par le formulaire sur une nouvelle ligne.

Après l'envoi du formulaire la page résultante sera celle du formulaire et le formulaire

devrait rappeler les informations envoyés (le message et le choix dans la liste déroulante devront être les mêmes que ceux envoyés). La dernière ligne du fichier texte (contenant les messages successivement envoyés par le formulaire) devra également être affichée sous le formulaire.

Vous ne devez pas hardcoder/écrire le formulaire directement en HTML. Vous devez utiliser le composant Form de Symfony ainsi que les types natifs de Symfony.

Chapitre VII

Exercice 03

	Exercice : 03
Exercice 03 : Cinquante nuances de couleurs.	
Dossier de rendu : *ex*03/	
Fichiers à rendre : `Tous les fichiers de l'application.`	
Fonctions Autorisées : `Toute functionnalité de Symfony.`	
Remarques : `n/a`	

Créer un nouveau bundle `E03Bundle` et référencez le dans le fichier AppKernel. Oubliez pas les routes.

Dans cet exercice vous devez créer une page que affiche un nombre de nuances pour les couleurs suivantes : *black*, *red*, *blue*, *green*. Le nombre de nuances devra être configurable dans le fichier `app/config/parameters.yml` dans le paramètre `e03.number_of_colors`.

La page devra contenir un header pour la tableau contenant les couleurs pour lesquelles les nuances sont affichées.

Les cases du tableau doivent avoir pour caractéristiques :

Hauteur : 40 pixels.

Largeur : 80 pixels.

Couleur de fond : une nuance de la couleur à laquelle correspond la colonne.

Le tableau devra afficher les nuances des 4 couleurs de telle façon à ce que l'on observe un dégradé sur autant de lignes que spécifiées dans le paramètre `e03.number_of_colors`.

Il est interdit de hardcoder les valeurs des couleurs en HTML, vous devez les créer dynamiquement et les envoyer au template Twig en tant que paramètre à partir du controller.