



Formation PHP - Symfony

D09 - One Page Apps using Symfony

Rares Serban rares.serban@pitechplus.com
42 Staff pedago@staff.42.fr

Résumé: Dans la suite de votre formations sur PHP Symfony, vous allez découvrir comment créer une application "Une page" avec Symfony, comment faire edes appels Ajax calls dans un navigateur et comment utiliser des Websockets pour la communication client-server.

Table des matières

I	Préambule	2
II	Consignes	3
III	Règles spécifiques de la journée	4
IV	Exercice 00	5
V	Exercice 01	6
VI	Exercice 02	7
VII	Exercice 03	8
VIII	Exercice 04	9

Chapitre I

Préambule

Aujourd'hui, vous allez apprendre comment créer une application "Une page" avec le framework Symfony.

SI vous ne savez pas ce qu'une application "one page", il s'agit tout simplement d'un site web qui ne nécessite pas que la page soit rechargée par le navigateur, tout se trouvant sur la page et le contenu étant mis à jour dynamiquement à l'aide d'appels Ajax à partir du code javascript.

On va aussi jeter un oeil aux sockets web qui permettent une communication dans les deux sens entre client et serveur à travers une connection TCP.

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 dépôt 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'exécuter avec `/bin/sh`.
- Vous ne devez laisser aucun autre fichier que ceux explicitement spécifiés par les énoncés des exercices dans votre dépôt 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

- Pour cette journée, votre repository doit contenir seulement une application Symfony.
- Vous devrez respecter les bonnes pratiques du framework Symfony.
- D'autres bundles tiers ou bibliothèques PEUVENT être utilisés.
- D'autres actions PEUVENT être créées dans les contrôleurs fournis, mais aucun autre contrôleur ne devrait être créé.
- Il est recommandé d'utiliser des services pour organiser votre code.

Chapitre IV

Exercice 00

	Exercice :
Exercice 00 : Bundle de base & entité de base	
Dossier de rendu : <i>ex/</i>	
Fichiers à rendre : Fichiers et répertoires de votre application	
Fonctions Autorisées : Toutes méthodes	
Remarques : n/a	

Pour cet exercice vous devez créer une application Symfony et un bundle personnalisé qui sera étendu au fur et à mesure des exercices. Le bundle s'appellera **D09Bundle** et aucun autre bundle devra exister dans votre application. Le répertoire du bundle ne devrait pas contenir d'autres répertoires inutiles.

Aussi, le répertoire **app/Resources/views** devrait être vide pour l'instant.

Créer une nouvelle entité dans le bundle nommée **Post**. Elle aura au moins les champs suivants. Le type de variables est intuitif à partir de leurs noms.


- id
- title
- content
- created

Le champ **created** prendra comme valeur la date à laquelle l'objet a été créé à l'aide des Doctrine Events.

Astuce : La commande **app/console doctrine :generate :entity** peut être utilisée pour générer des entités.

Chapitre V

Exercice 01

	Exercice :
Exercice 01 : Formulaire nouvelle publication & Sécurité	
Dossier de rendu : <i>ex/</i>	
Fichiers à rendre : Fichiers et répertoires de votre application	
Fonctions Autorisées : Toutes méthodes	
Remarques : n/a	

Seulement les utilisateurs identifiés doivent pouvoir créer des publications (posts). Pour ce faire, mettre en place la sécurité nécessaire et créer quelques utilisateurs simples. Créer un contrôleur **UserController** avec une action **loginAction** pour la route **/login**. Cette action doit retourner un template **login.html.twig** contenant un formulaire de login. Le formulaire sera envoyé via Ajax.

Créer aussi un contrôleur **PostController** avec une action **defaultAction** pour le chemin par défaut qui retournera un template appelé **index.html.twig**. Le template devra soit afficher le formulaire de login si l'utilisateur n'est pas connecté soit un formulaire de création de nouveau post si il est déjà connecté.

Le formulaire de création de nouveau post devra contenir les champs **title** et **content** et sera créé en utilisant une classe de type Post.


Le formulaire de login doit être envoyé via Ajax et si les identifiants sont corrects on affichera directement le formulaire de création de nouveau post. Si les identifiants ne sont pas corrects on affichera une alerte dans le navigateur.

Le formulaire de création de nouveau post sera également envoyé via Ajax et afficher un message de confirmation si le post a été correctement ajouté.

Astuce : *Symfony fournit une classe JsonResponse de base, des handlers d'authentification et la possibilité de faire des sous-requêtes à partir de templates. Assurez-vous d'avoir sécurisé les actions de votre contrôleur !*

Chapitre VI

Exercice 02

	Exercice :
Exercice 02 : Affichage de la liste des publications	
Dossier de rendu : <i>ex/</i>	
Fichiers à rendre : Fichiers et répertoires de votre application	
Fonctions Autorisées : Toutes méthodes	
Remarques : n/a	

Créer une liste basique sous le formulaire de la page principale. La liste devra afficher tous les posts créés avec leur titre et date de création. La liste peut être vue même si aucun utilisateur n'est connecté/authentifié.


Modifier la soumission du formulaire de telle sorte que cette liste soit mise à jour avec les nouveaux posts créés à la fin de la liste au moment où l'appel Ajax est terminé.

Aussi, implémenter la validation du formulaire afin de s'assurer que le titre de chaque post est unique. Si on essaie d'ajouter un nouveau post ayant un titre qui existe déjà dans un autre post, vous devez afficher un message d'erreur dans une alerte du navigateur et (bien sûr) ne pas effectuer l'insertion.

Astuce : *Il est fortement recommandé d'utiliser des traductions pour les libellés des champs du formulaire et les messages d'erreur. Et rappelez-vous, la page web ne doit jamais être rechargée !*

Chapitre VII

Exercice 03


	Exercice :
Exercice 03 : Details de publication & Supression	
Dossier de rendu : <i>ex/</i>	
Fichiers à rendre : Fichiers et répertoires de votre application	
Fonctions Autorisées : Toutes méthodes	
Remarques : n/a	

Créer une action dans **PostController** appelée **viewAction** avec la route **/view/{id}** qui sera appelée par Ajax lors du click sur le titre d'un post et devra retourner tous les détails du post. Les détails seront affichés au dessus du formulaire.

Créer une autre action dans le même controller appelée **deleteAction** avec la route **/delete/{id}** qui, lorsqu'elle sera appelée avec Ajax, supprimera le post. Seulement les utilisateurs identifiés devrait pouvoir supprimer un post. Les détails du post devraient contenir un bouton *Delete* qui devrait demander une confirmation par une boite de dialogue avant de supprimer le post. Cette même action devra mettre à jour la liste des posts en enlevant le post supprimé de cette liste.

Chapitre VIII

Exercice 04

	Exercice :
Exercice 04 : Websockets !	
Dossier de rendu : <i>ex/</i>	
Fichiers à rendre : Fichiers et répertoires de votre application	
Fonctions Autorisées : Toutes méthodes	
Remarques : n/a	

Il est temps d'apprendre ce que c'est que les Websockets. Si vous n'en avez jamais entendu parler c'est le moment de les découvrir !

Premièrement vous allez mettre en place le serveur Websocket sur Symfony sur le port 8080. Le serveur démarera à l'aide d'une commande comme suit :

app/console websocket :server

Du côté client, vous devrez créer un nouveau Websocket à l'aide de Javascript. Maintenant, chaque fois qu'un post est ajouté dans n'importe quelle fenêtre du navigateur, vous devrez mettre à jour la liste des posts avec le nouveau post ajouté dans tous les clients connectés (donc dans toutes les fenêtres affichant votre application).

Assurez vous de modifier également le formulaire de suppression de post dans toutes les fenêtres du navigateur.

Astuce : *Cet exercice peut être résolu de multiples façons. Une variété de bibliothèques peuvent être trouvées en ligne qui peuvent vous aider à implémenter la plupart de ces fonctionnalités.*