TP1 symfony

Objectifs

Créer un premier site avec le framework symfony 5 en se concentrant uniquement sur le contrôleur et la vue à travers le moteur de template twig.

Durée

3h

Préparation du poste de travail

Vous allez utiliser **les machines virtuelles linux** pour réaliser l'ensemble des Tps de ce semestre.

La création d'un projet symfony se fait en ligne de commande.

Placez-vous dans le dossier /media/Qi/$USER et tapez la commande suivante :

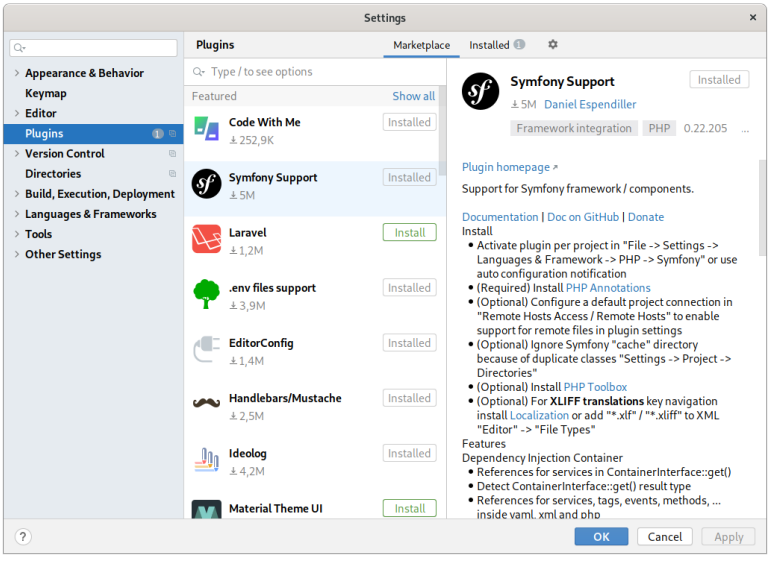
```
composer create-project symfony/skeleton tp1Symfony
```

Nous utiliserons phpstorm comme IDE pour développer nos sites. Vous trouverez PhpStorm dans les outils installés.

PhpStorm est un outil commercial. En tant qu'étudiant, vous pouvez obtenir une licence gratuite en vous inscrivant avec votre adresse universitaire sur le site de Jetbrain.

Il permet d'utiliser un terminal intégré ainsi qu'un connecteur sur les bases de données. (Et pleins d'autres choses …)

Ouvrez le dossier tp1Symfony avec phpStorm. Installez le plugin Symfony pour PhpStorm :



Nous utiliserons maintenant que le terminal intégré à phpStorm.

Nous allons installer des composants additionnels pour notre Tp.

```
composer require --dev maker profiler
```

```
composer require annotations twig form validator
```

Nous pouvons maintenant lancer notre site grâce à la commande symfony :

```
symfony server:start
```

Normalement, si vous allez sur http://127.0.0.1:8000, vous devez voir votre site. Vous tombez sur une page 404 de symfony. (La page demandée n'est pas trouvée.)

Première page

Nous voulons réaliser le site présent sur cette page :
https://ntrugeon.lpmiaw.univ-lr.fr/tp1symfony2020/symfony/public/

Vous trouverez sous moodle l'ensemble des éléments nécessaires à la réalisation de ce site.

Copiez les dossiers images, css, js, fonts dans public/

Créer un nouveau controller nommé DefaultController en utilisant la bonne commande symfony.

```
bin/console make:controller
```

Vous pouvez vérifier qu'il est présent en ajoutant dans l'URL /default/

La fonction index du controller utilise la fonction render qui va appeler une page twig (la vue) en lui passant en paramètre un tableau qui contient une variable controller_name qui vaut DefaultController. **Cette variable n'est utile que pour montrer comment passer une variable à la vue.**

Changez la valeur de cette variable de DefaultController en variableAAfficher.

Modifiez l'annotation du controller pour que la route soit « / » et modifiez le nom de la route pour qu'elle ne nomme « homepage »

Modifiez la vue se situant dans template/default/index.html.twig en y plaçant de manière adéquate le code html fourni. Vous pouvez donc supprimer du contrôleur la variable controller_name.

Cela fonctionne, mais nous travaillons en local et nous ne savons pas comment sera hébergé notre code. Nous allons rendre portable les ressources (images, css, js, etc.) à travers les assets dans twig.

```
composer require asset
```

Modifiez l'ensemble des ressources externes en ajoutant la fonction asset :
{{ asset('images/logo.png') }} pour une image

Pour l'instant le formulaire n'est pas en place, nous verrons cela un peu plus tard.

À ce stade, vous devez avoir les images qui apparaissent, la mise en forme du site qui est correcte et la partie responsive qui fonctionne.

Pour tester que tout fonctionne bien, modifier la route par défaut en /test/validation.

Deuxième page

Ajoutez sur le même modèle, une seconde méthode à votre contrôleur avec comme route /produits qui permet d'afficher la page produits.html du modèle.

Créez la page twig correspondante. Vous aurez besoin d'utiliser la notion de path dans twig pour que vos liens du menu fonctionnent.

Passage de paramètre

Symfony permet facilement de passer des paramètres à une page à travers des URI simples et pratiques. Par exemple, la page /produits/6 correspond au produit numéro 6, /page/ma-page correspond à la page qui se nomme ma-page. https://en.wikipedia.org/wiki/Semantic_URL

Pour cela on va modifier l'annotation de l'action produits dans le contrôleur en mettant :

```
@Route("/produits/{id}",requirements={"id": "[1-9]\d*"}, name="produits")
```

Cela veut dire qu'on peut rajouter un identifiant et uniquement un numéro à la suite de la page produits.

Il suffit de le passer en paramètre à la vue sous forme de tableau. N'oubliez pas d'ajouter un paramètre $id à votre méthode.

Faites en sorte de concaténer le numéro du produit au titre h3 de la page.

Ensuite, sur la page d'accueil, il y a 3 produits. Faites en sorte, qu'on puisse afficher l'image correspondante au produit pour chaque détail : « Learn more ».

Troisième page

Créez une troisième page de votre choix qui permet de faire fonctionner le lien « histoire » du menu. Je vous laisse faire pour créer tous les éléments nécessaires.

Formulaire

Pour l'instant, notre formulaire est écrit en html mais n'est pas fonctionnel.

Nous allons le transformer à la mode symfony pour permettre qu'il puisse afficher un message et simuler l'envoi d'un mail. Il faut tout d'abord installer le composant Mailer

```
composer require mail
```

Par défaut, en mode de développement, les mails ne seront pas envoyés.

https://symfony.com/doc/current/forms.html

Si nous voulons que les messages soient stockés dans la base de données, il faudrait créer une classe Message pour conserver l'ensemble des informations. Nous ne voulons faire que l'envoi effectif du message.

Avec une commande symfony, créez un formulaire appelé ContactType.php. (Les fichiers suffixés de Type dans symfony sont des formulaires.). Ils sont placés dans le dossier Form.

```
bin/console make:form ContactType
```

Ouvrez le fichier et observez les méthodes associées :

La fonction buildForm permet de créer votre formulaire. Vous utiliserez la méthode add associée à l'objet builder pour ajouter des champs à votre formulaire.

Exemple de la méthode add :

```
$builder->add('name', TextType::class, array('attr' => array('placeholder' => 'Votre nom'), 'constraints' => array(new NotBlank(array("message" => "Merci de renseigner votre nom."))))
```

La méthode configureOptions permet de spécifier des options par défaut.

La vue

Dans le fichier de vue, il faut maintenant supprimer notre formulaire fictif pour le remplacer par notre formulaire symfony.

Pour cela, il suffit d'ajouter ce code :

```
{{ form_start(form) }}
{{ form_widget(form) }}
<button type="submit">Envoyez</button>
{{ form_end(form) }}
```

Le controlleur

Reste à dire au contrôleur de gérer le formulaire :

La méthode index du contrôleur prend un objet Request et un objet \Swift_Mailer en paramètre. Ces objets permettent de savoir comment a été appelé notre page (POST, GET, …) et d'envoyer des emails.

On se trouve dans la classe DefaultController qui hérite de la classe Controller. Nous pouvons utiliser l'ensemble des méthodes de cette classe :

createForm : permet de créer le formulaire à partir de la classe Type précédemment créée. Cela retourne un objet formulaire.

render : permet d'appeler la vue twig en lui passant des paramètres.

redirectToRoute : permet de rediriger le site vers une nouvelle route. Équivalent à header().

Votre objet formulaire peut utiliser les méthodes suivantes :

handleRequest : prends en paramètre l'objet Request et permet de traiter le formulaire.

isValid : renvoie vrai si le formulaire est correct en fonction des contraintes qu'on lui a donné ou faux dans le cas contraire.

IsSubmitted : pour savoir s'il a bien été soumis.

Je vous donne la fonction sendMail qui va bien à ajouter au contrôleur :

https://symfony.com/doc/current/mailer.html

```
private function sendEmail($data, MailerInterface $mailer){
     $email = new EmailMailer();
     $email->from($data["email"])
       ->to('nicolas.trugeon@univ-lr.fr')
       ->subject($data["nom"])
       ->text($data["message"]);
     $mailer->send($email);
}
```

Algorithme à reproduire pour l'action du contrôleur :

On crée le formulaire
on traite le formulaire
si le formulaire est soumis et qu'il est valide
    on envoie le mail avec les données
    on retourne une page de redirection
on retourne la vue avec le formulaire

En local, la fonction d'envoi de mail n'est pas active mais sur un vrai serveur, on peut facilement le configurer en modifiant le fichier .env.local