

Table des matières

[1. Introduction. 3](#_Toc12371945)

[1.1Quelques logiciels utiles pour fonctionner avec Symfony . 3](#_Toc12371946)

[2. C’est parti. 3](#_Toc12371947)

[2.2 Vérifier la version de PHP : 3](#_Toc12371948)

[2.3 Vérifier la version de composer : 4](#_Toc12371949)

[3. Installation 4](#_Toc12371950)

[3.1 Créer un projet 4](#_Toc12371951)

[4. Les controllers 6](#_Toc12371952)

[5. Création des routes. 7](#_Toc12371953)

[6. Création d’une vue. 7](#_Toc12371954)

[7. Parlons Twigg 8](#_Toc12371955)

[8. Affichage du site. 9](#_Toc12371956)

[9. Vue d’un article 11](#_Toc12371957)

[10. Du bouton vers la vue 11](#_Toc12371958)

[11. Création de la page vue d’un article 12](#_Toc12371959)

[12. L’ORM Doctrine 12](#_Toc12371960)

[13. Les lignes de commandes pour doctrine 14](#_Toc12371961)

[14. Accès aux données. 15](#_Toc12371962)

[15. Variable dans une route. 17](#_Toc12371963)

[16. Injection de dépendance… 17](#_Toc12371964)

[17. Astuces 17](#_Toc12371965)

Tuto 1/4 , blog :

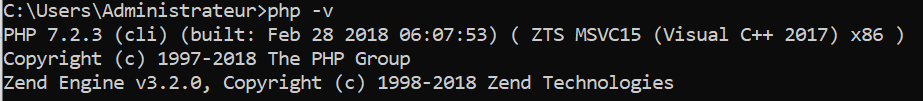
1. Introduction.

1.1Quelques logiciels utiles pour fonctionner avec Symfony .

* Wamp : nécessaire pour faire tourner S4, car il embarque PHP 7.2.
  + <http://www.wampserver.com/>
* Composer : gestionnaire de libraire et de dépendances pour PHP.
  + <https://getcomposer.org/>
* Et bien sur Symfony.
  + <https://Symfony.com/download>
* IDE : Php Storm ou VisualStudioCode ou comme tu veux.
* Git, idéalement avec Gitkraken pour la belle interface.
* Installer Yarn pour la gestion des bootstrap.
* Installer Twig : composer require "twig/twig:^2.0"

1. C’est parti.
   1. Vérifier la version de PHP :

Pour cela lancer une commande < CMD> en mode administrateur :



La commande est PHP –v, ici la version est 7.2.3.

* 1. Vérifier la version de composer :

Pour cela lancer la commande : 

Ici la version de Composer est 1.8.6.

1. Installation

3.1 Créer un projet

Ouvrir une invite de commande, se placer dans le dossier pour le projet. Il est n’est pas nécessaire de se mettre dans le dossier « www » ou « htdocs », car il y a un serveur interne.





La commande est : composer create-projet symfony/website-skeleton blog (blog étant le nom du projet ^^).

* 1. Le serveur

Symfony fonctionne avec des lignes de commande autant passé par le terminal fournit dans l’iDE.

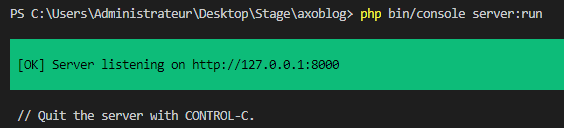


Avec la commande suivante je passe par composer pour lui demander un serveur :

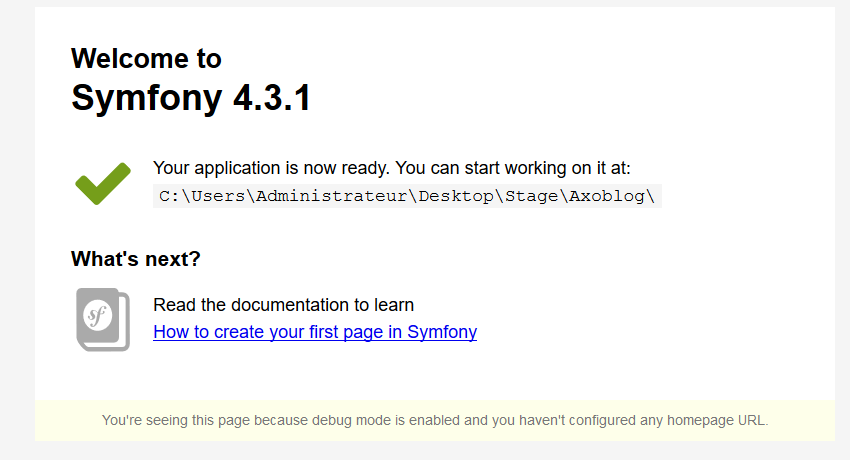
composer require server –dev

-dev : ici pour préciser le mode développeur, on pourra le modifier ensuite pour passer en prod.





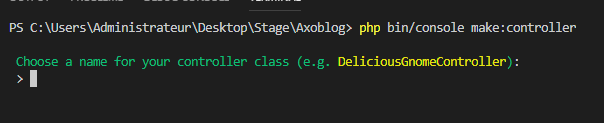
Le serveur est lancé et la page est accessible.



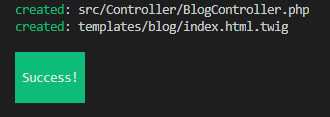
1. Les controllers



La commande est php bin/console make :controller



Et choisir le nom du controller ici BlogController



On peut tester la route du controller « /blog » et vérifier si cela fonctionne.

1. Création des routes.

Pour changer la route de la page home :

/\*\*

\*

\* @Route("/",name="home")

\*

\* @return void

\*/

public function home ()

{

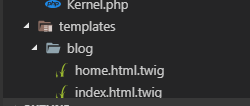
return $this -> render('blog/home.html.twig');

}

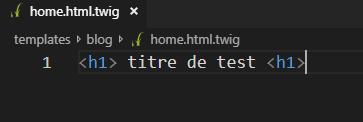
Le « / » seul permet d’amener la personne qui se connecte directement sur la page voulue. A présent il faut configurer la vue afférente.

1. Création d’une vue.

Copier la vue « index.html.twig » et coller sur le dossier « blog ».



Insérer un titre dans la vue pour vérifier son fonctionnement.



Se connecter en 127.0.0.1 :8000



Et voilà le travail.



1. Parlons Twigg

Pour info c’est la même société que Symfony qui gère Twigg.





Exemple :

{% if age > 18 %}

<p> Tu es majeur </p>

{% else %}

<p> Tu es mineur </p>

{% endif %}

En passant âge en paramètre dans le controller.

1. Affichage du site.



Dans la vue « base » il suffit d’insérer le lien vers le template choisit, par exemple :

<title>{% block title %}Welcome!{% endblock %}</title>

<link rel="stylesheet"

href:"https://bootswatch.com/4/flatly/bootstrap.min.css">

{% block stylesheets %}{% endblock %}

Ajout d’une nav barre :

<nav class="navbar navbar-expand-lg navbar-dark bg-primary">

<a class="navbar-brand" href="#">Navbar</a>

<button class="navbar-toggler" type="button" data-toggle="collapse" data-target="#navbarColor01" aria-controls="navbarColor01" aria-expanded="false" aria-label="Toggle navigation">

<span class="navbar-toggler-icon"></span>

</button>

<div class="collapse navbar-collapse" id="navbarColor01">

<ul class="navbar-nav mr-auto">

<li class="nav-item active">

<a class="nav-link" href="#">Home <span class="sr-only">(current)</span></a>

</li>

<li class="nav-item">

<a class="nav-link" href="#">Features</a>

</li>

<li class="nav-item">

<a class="nav-link" href="#">Pricing</a>

</li>

<li class="nav-item">

<a class="nav-link" href="#">About</a>

</li>

</ul>

<form class="form-inline my-2 my-lg-0">

<input class="form-control mr-sm-2" type="text" placeholder="Search">

<button class="btn btn-secondary my-2 my-sm-0" type="submit">Search</button>

</form>

</div>

</nav>

Modification de la navbar ainsi que des routes «  href = ‘#’ » en passant les vues en lien :

<nav class="navbar navbar-expand-lg navbar-dark bg-primary">

<a class="navbar-brand" href="/">Axoblog</a>

<button class="navbar-toggler" type="button" data-toggle="collapse" data-target="#navbarColor01" aria-controls="navbarColor01" aria-expanded="false" aria-label="Toggle navigation">

<span class="navbar-toggler-icon"></span>

</button>

<div class="collapse navbar-collapse" id="navbarColor01">

<ul class="navbar-nav mr-auto">

<li class="nav-item ">

<a class="nav-link" href="/blog">Articles <span class="sr-only">(current)</span></a>

</li>

<li class="nav-item">

<a class="nav-link" href="/blog/new"> Creer un article</a>

</li>

</ul>

</nav>

Enfin pour que tous les contenus soit inclus dans le template :

<div> class="container"

{% block body %}{% endblock %}

</div>



Ensuite on appelle la vue « base » dans la vue qui nous intéresse, ici home.html.twig.

{% *extends* 'base.html.twig' %}

1. Vue d’un article

Une fois la vue prête (pas de tuto sur le sujet), on ajoute dans le contrôleur la possibilité de voir un seul article.

*/\*\*  
 \** ***@Route****("blog/12", name="blog\_show")  
 \*/  
public function* show(){  
 *return* $this -> render ('blog/show.html.twig');  
  
}

Pour l’instant l’id est défini manuellement, on le rendra dynamique par la suite.

1. Du bouton vers la vue

Ajout du chemin vers l’article concerné :

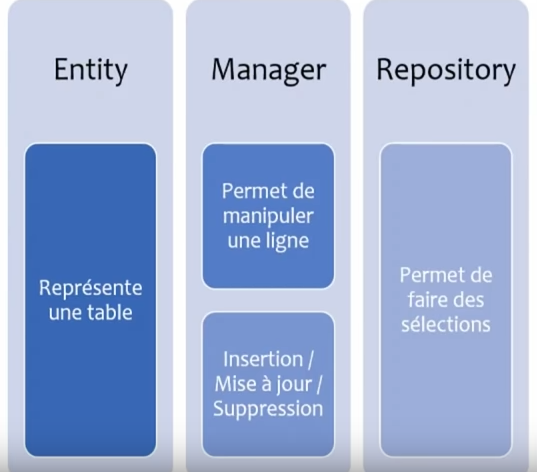
<a *href*="{{ path ('blog\_show') }}" *class*="btn btn-primary">Lire la suite</a>

1. Création de la page vue d’un article

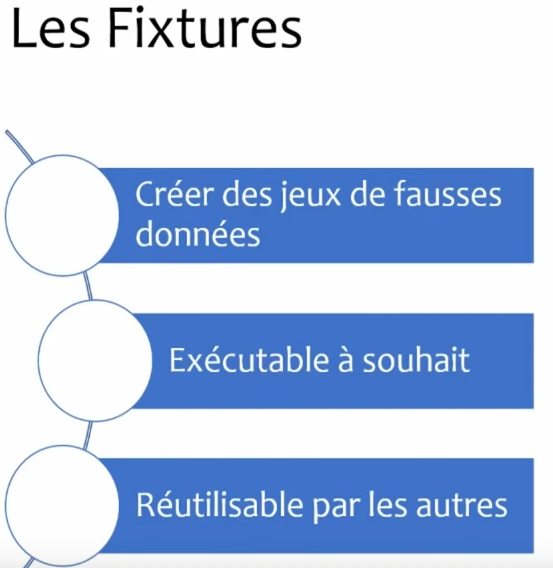
Dans les template /blog , créer un vue show.html.twig.

Il suffit de faire un copier-coller d’un article et de le remettre dans la vue.

1. L’ORM Doctrine



Notions intéressantes les fixtures, voici le schéma :



Dans l’arborescence du projet, dans le dossier .env se trouvent les renseignements nécessaire à la gesiton de la BDD.

*DATABASE\_URL*=mysql://db\_user:db\_password@127.0.0.1:3306/db\_name

Modification :

*DATABASE\_URL*=mysql://root:@127.0.0.1:3306/Axobdd

1. Les lignes de commandes pour doctrine

* Création de la BDD : php bin/console doctrine:database:create



L’entité crée sera injecter dans la BDD comme une table, appelons la « article ».

Ensuite il faut intégrer les données dans la BDD.



Les fichiers migrations permettent de noter tous les changements qui ont lieu dans les entités et permettent aux dev qui téléchargent le projet de mettre leur bdd à jour via une migration.



Maintenant ajoutons des données dans la base avec fixtures :

D’abord télécharger le pack fixtures :

composer require orm-fixtures

ensuite :

php bin/console make:fixtures

On complete l’ArticleFixture :



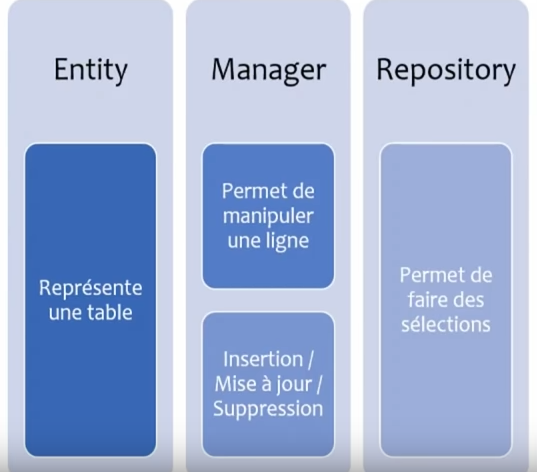
composer require fzaninotto/faker

*class* ArticleFixtures *extends* Fixture  
{  
 *public function* load(ObjectManager $manager)  
 {  
 *for* ($i=0; $i<=10; $i++){  
 $article = *new* Article();  
 $article ->setTitle("titre de l'article n°$i")  
 -> setContent("<p> contenu de l'article n°$i</p>")  
 -> setImage("http://placehold.it/350x150")  
 -> setCreatedAt(*new* \DateTime());  
 $manager->persist($article);  
  
  
 }  
 $manager->flush();  
 }  
}



1. Accès aux données.

Pour rappel :





*// affichage de tous les articles  
/\*\*  
 \** ***@Route****("/blog", name="blog")  
 \*/  
public function* index()  
{  
 $repo = $this ->getDoctrine()->getRepository(Article ::*class*);  
 $articles =$repo ->findAll();  
 *return* $this->render('blog/index.html.twig', [  
 'controller\_name' => 'BlogController',  
 'articles' => $articles  
 ]);  
}

Rendre visible tous les articles dans la page /blog :

{% *extends* 'base.html.twig' %}  
{% *block* title %}Hello Axobloger!{% *endblock* %}  
  
{% *block* body %}  
{% *for* article *in* articles %}  
 <article>  
 <h2 >{{ article.title }} </h2>  
 <div *class*="metadata"> Ecrit le {{ article.createdAt | date('d/m/Y') }} à {{ article.createdAt | date ('H:i') }}</div>  
 <div *content*>  
 <img *src*='{{ article.image }}' *alt*="">  
 {{ article.content | raw }}  
 </div>  
 <a *href*="{{ path ('blog\_show') }}" *class*="btn btn-primary">Lire la suite</a>  
 </article>  
{% *endfor* %}  
{% *endblock* %}

J’ai utilise un filtre twig pour afficher la date :

Et un autre pour afficher le contenu “content” sans modification avec |raw



1. Variable dans une route.

*//affichage d'un seul article  
/\*\*  
 \** ***@Route****("blog/{id}", name="blog\_show")  
 \*/  
public function* show($id){  
 $repo = $this ->getDoctrine()->getRepository(Article ::*class*);  
 $article = $repo -> find($id);  
 *return* $this -> render ('blog/show.html.twig',[  
 'article'=$article  
 ]);  
  
  
}

Dans la page Index.html.twig,

Il est nécessaire de modifier la route du bouton :

<a *href*="{{ path('blog\_show', {'id': article.id}) }}" *class*="btn btn-primary">Lire la suite</a>

Avec path on peut passer le nom de la route , et la valeur d’une variable.

1. Injection de dépendance…



*// affichage de tous les articles  
/\*\*  
 \** ***@Route****("/blog", name="blog")  
 \*/  
public function* index(ArticleRepository $repo )  
{  
 $articles =$repo ->findAll();  
 *return* $this->render('blog/index.html.twig', [  
 'controller\_name' => 'BlogController',  
 'articles' => $articles  
 ]);  
}

Ici on passe le repo dans les paramètres de la fonction.

1. Astuces

-Cls : pour nettoyer l’écran du terminal.

-Alt + F12 : pour ouvrir un terminal dans php storm

-Permet d’inserer des images et de simuler leurs présences. Ex : <img *src*="http://placehold.it/350x150" *alt*="">

- le lien vers le tuto : https://www.youtube.com/watch?v=UTusmVpwJXo