# - SYMFONY INSTALLATION ET DEMARRAGE -

- installer git

- installer php (via xamp ou autre)

- installer composer

- https://symfony/installer (celui-ci télécharge symfony.phar)

- copier ce fichier dans un dossier de notre projet

- mettre php dans les variables d'environnement

- Sur la console, taper la commande :

> php symfony.phar

si erreur suivante :

symfony ssl certificate problem unable to get local issuer certificate

Alors aller sur ce lien :

https://stackoverflow.com/questions/37997669/curl-error-60-ssl-certification-issue-when-attempting-to-use-symfony

(

Follow this link, Copy the entire page and save it in a: "cacert.pem"

Then in your php.ini file insert or edit the following line: curl.cainfo = "[pathtothisfile]\cacert.pem"

Or

use composer instead Symfony installer

)

# CREATION PROJET ET LANCEMENT SERVEUR

- Pour créer un nouveau projet :

> php symfony.phar new premier-projet

- pour lancer le server symfony, se placer dans le dossier de notre projet et taper :

> bin/console server:run

Remarque: dans bin/console se trouve toutes les commandes console de symfony

# ROUTE ET ACTION DANS LE CONTROLLER --

Il est possible de définir les routes dans les annotations de l'action.

exemple:

use Sensio\Bundle\FrameworkExtraBundle\Configuration\Route;

/\*\*

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

\*/

public function indexAction(Request $request){}

\* @Route(

\* "/hello/{name}/{age}",

\* defaults={"name": "World", "age": 0},

\* requirements={"age":"\d{1,3}", "name":"\D+"}

\* )

public function helloAction($name) {return new Response('hello ' . $name);}

Il est aussi possible de mettre plusieurs routes qui vont réagir à la même action

il suffit d'ajouter une autre annotation pour la 2e route.

Exemple

/\*\*

\* @Route() route 1

\* @Route() route 2

\*

\*/

# PLUGIN DANS PHPSTORM

- FRAMEWORK INTEGRATION

Ensuite, Aller dans settings->framework et php->symfony, cocher Enable...

# LES VIEWS

Par defaults, on met les views dans app/resources/[dossier du controller]/view.html.twig

Mais si on pense créer un bundle réutilisable, on mettra les vues dans le dossier de notre

Bundle.

# CONTROLLER CREATION

Pour créer un controller:

$ bin/console generate:controller

Exemple :

$ bin/console generate:controller

Welcome to the Symfony controller generator

Every page, and even sections of a page, are rendered by a controller.

This command helps you generate them easily.

First, you need to give the controller name you want to generate.

You must use the shortcut notation like AcmeBlogBundle:Post

Controller name: AppBundle:Calculator

Determine the format to use for the routing.

Routing format (php, xml, yml, annotation) [annotation]:

Determine the format to use for templating.

Template format (twig, php) [twig]:

Instead of starting with a blank controller, you can add some actions now. An action

is a PHP function or method that executes, for example, when a given route is matched.

Actions should be suffixed by Action.

New action name (press <return> to stop adding actions): addAction

Action route [/add]: /add/{n1}/{n2}

Template name (optional) [AppBundle:Calculator:add.html.twig]: :calculator:add.html.twig

New action name (press <return> to stop adding actions):

Summary before generation

You are going to generate a "AppBundle:Calculator" controller

using the "annotation" format for the routing and the "twig" format

for templating

Do you confirm generation [yes]?

Controller generation

created .\src\AppBundle/Resources/views/calculator/

created .\src\AppBundle/Resources/views/calculator/add.html.twig

created .\src\AppBundle/Controller/CalculatorController.php

created .\src\AppBundle/Tests/Controller/

created .\src\AppBundle/Tests/Controller/CalculatorControllerTest.php

Generating the bundle code: OK

Everything is OK! Now get to work :).

# ROUTING

Connaitre toutes les routes de l'application:

> php bin/console debug:router

ou

> php bin/console debug:router --env=prod

ou

> php bin/console debug:router --env=prod --show-controllers

pour l'exporter vers un fichier,

> php bin/console debug:router > routes.txt

# GESTION DES DEPENDANCES JS - BOWER

Installer NodeJS puis installer BOWER

> npm install -g bower

> bower init

Dans la racine du premier-projet, créer un fichier .bowerrc

dans le fichier, écrire

{

"directory": "web/bower\_assets"

}

> bower install bootstrap --save

# DOCTRINE

## Intro

DOCTRINE est un ORM (Object Relational Mapper)

DBAL (Database Access Layer)

1ere chose à faire pour utiliser DOCTRINE

configurer le fichier : app/config/parameters.yml

parameters:

database\_host: 127.0.0.1

database\_port: null

database\_name: symfony

database\_user: root

database\_password: null

Utilisation de plusieurs BDD dans DOCTRINE, voir lien :

http://www.dinduks.com/symfony2-utiliser-plusieurs-bases-de-donnees/

Etape 1:

Définir les entités.

Exemple:

class Auteur {

private $id;

private $surname;

private $name;

private $email;

private $password;

// Avec les guetters et les setters

}

## Création BDD, Entité, Getters et Setters

Pour générer les entités et les getters et setters

Commande console

php bin/console doctrine:generate:entity

php bin/console doctrine:generate:entities <nomClass>

Activer : extension=php\_pdo\_mysql.dll

Démarrer: mysql

Pour créer la BDD et/ ou une entité:

> php bin/console doctrine:database:create

Pour générer les entités

> php bin/console doctrine:generate:entity

et réponse aux questions

(>php bin/console doctrine:generate:entities <nomClass>) : Cette commande permet de générer des getters et setters automatiquement

Dans les Entity se trouveront la définition de nos classes

Dans les class Repository c'est la que se trouveront nos requetes

Pour créer le Schema de la bases-de-donnees

> php bin/console doctrine:schema:create

si modification sur le schema, faire > php bin/console doctrine:schema:update –force

Pour persister les données : il faut un Entity Manager

Exemple :

// Création d'une entité book  
$book = **new** Book();  
$book->setTitle("SQL for smarties")  
 ->setAuthor('Joe Celko')  
 ->setPrice(54.8)  
 ->setPublishedAt(**new** \DateTime("200-01-25"));  
  
// Récupération d'une instance du gestionnaire d'entité  
$em = $this->getDoctrine()->getManager();  
  
// Persistance de l'entité  
$em->persist($book);  
  
// Validation de la transaction  
$em->flush();

Pour Récupérer les données et faire une modification

Pour la récupération, il faut : Entity Repository

Exemple :

// Récupération de data avec Entity Repository  
$repository = $this->getDoctrine()  
 ->getRepository("AppBundle:Book");  
$book = $repository->find(2);  
  
// Modification  
$book->setTitle("Advanced SQL");  
  
// Persistance de data avec Entity Manager  
$em = $this->getDoctrine()->getManager();  
$em->persist($book);  
$em->flush();

## Les relations

### ManyToOne : PLUSIEURS(many) livres peuvent être édités par UN(one) publisher

Remarque: ***@ORM\JoinColumn****(nullable=false)* veut qu’on interdit de null sur cette colonne,

*cascade={"persist"}*

veut dire quand on enregistre un livre, il persiste aussi un editeur

**class** Book  
{  
 */\*\*  
 \** ***@var*** *Publisher  
 \** ***@ORM\ManyToOne****(targetEntity="AppBundle\Entity\Publisher", inversedBy="books", cascade={"persist"})  
 \** ***@ORM\JoinColumn****(nullable=false)  
 \*/* **private** $publisher;  
  
}

### OneToMany: UN(one) publisher peut publier PLUSIEURS(many) livres

Remarque:

Quand on a un OneToMany, cette relation ne fonctionne pas toute seule, il faut faire de l’autre côté (2e Class) un OneToMany et dans la 1ere Class, faire un mappedBy vers la variable de la 2e Class et cette variable doit avoir un ManyToOne et un inversesBy pointant vers la variable de la 1ere Class.

Exemple (1ere Class : Publisher)

**class** Publisher  
{  
 */\*\*  
 \** ***@var*** *ArrayCollection  
 \** ***@ORM\OneToMany****(targetEntity="AppBundle\Entity\Book", mappedBy="publisher")  
 \*/* **private** $books;  
}

* **php bin/console doctrine:generate:entities AppBundle:Publisher**

Cette commande permet de générer des getters et setters automatiquement après avoir fait les annotations pour les relations.

Exemple final :

**class** Publisher  
{  
 */\*\*  
 \** ***@var*** *ArrayCollection  
 \** ***@ORM\OneToMany****(targetEntity="AppBundle\Entity\Book", mappedBy="publisher")  
 \*/* **private** $books;  
}  
  
**class** Book  
{  
 */\*\*  
 \** ***@var*** *Publisher  
 \** ***@ORM\ManyToOne****(targetEntity="AppBundle\Entity\Publisher", inversedBy="books", cascade={"persist"})  
 \** ***@ORM\JoinColumn****(nullable=false)  
 \*/* **private** $publisher;  
  
 */\*\*  
 \** ***@var*** *ArrayCollection  
 \** ***@ORM\ManyToMany****(targetEntity="AppBundle\Entity\Author", inversedBy="books", cascade={"persist"})  
 \*/* **private** $authors;  
}  
  
**class** Author  
{  
 */\*\*  
 \** ***@var*** *ArrayCollection  
 \** ***@ORM\ManyToMany****(targetEntity="AppBundle\Entity\Book", mappedBy="authors")  
 \*/* **private** $books;  
  
}

php bin/console doctrine:generate:entities AppBundle:Book

php bin/console doctrine:generate:entities AppBundle:Publisher

php bin/console doctrine:generate:entities AppBundle:Author

Voir lien :

<http://docs.doctrine-project.org/projects/doctrine-orm/en/latest/reference/association-mapping.html>

**Remarque :**

**Si je veux créer une table de jointure qui a d’autres champs, il faut donc créer une Entité de jointure**

**Et faire des OneToMany et ManyToOne avec les 2 autres Classes.**

## LifeCycleCallbacks : Gérer les évènements

@ORM\HasLifecycleCallbacks() à mettre en début de la class

Exemple :

/\*\*

\* Auteur

\*

\* @ORM\Table(name="auteurs")

\* @ORM\Entity

\* @ORM\HasLifecycleCallbacks()

\*/

class Auteur

{

...

}

Une fois que c’est fait, il faut créer une méthode :

PrePersist, PostPersist, PreUpdate, PostUpdate, PreRemove, PostRemove, PreFlush, PostLoad

Voir exemple dans le code.

## Composant de validation

Voir la doc :

<https://symfony.com/doc/current/validation.html#constraints>

Faire un use de la class Contraints, Puis dans les annotations de la propriétés ou methods, faire @Assert\<la contrainte>

Exemple :

use Symfony\Component\Validator\Constraints as Assert;

class Auteur

{

/\*\*

\* @var string

\*

\* @Assert\NotBlank()

\*

\* @ORM\Column(name="nom", type="string", length=30)

\*/

private $nom;

}