# Symfony

## Squelette projet Symfony :

public/ -> Tous ce qui sera visible par l’utilisateur : les images , les js, les pages html, le css…

src/ -> les classes ( controller et models), notre code php, dans les controllers chaque methode est un controller avec sa route et sa route-name

vendor/ -> ne pas toucher ,la où sont installer les recettes (=bundles) par composer

var/ -> le cache générer

bin/ -> bin : pas utile on vient l' exécuter avec bin/console, la où il y a un fichier php pour generer du php

## Twig = moteur de template pour les vues :

* {%%} = balise twig ( {%set var = val%} )
* {# un commentaire #}
* {{ var }} = interpolation (affichage de var)
* {{ tabl | method()|methode2 }} = exécute les méthodes method et methode 2 sur la variable tabl

## Les formulaires sur twig :

* form\_start() affiche la balise d'ouverture du formulaire HTML, soit<form>. Il faut passer la variable du formulaire en premier argument, et les paramètres en deuxième argument. L'index attr des paramètres, et cela s'appliquera à toutes les fonctions suivantes, représente les attributs à ajouter à la balise HTML générée, ici le<form>. Il nous permet d'appliquer une classe CSS au formulaire, ici form-horizontal.
* form\_errors() affiche les erreurs attachées au champ donné en argument. Nous verrons les erreurs de validation dans le prochain chapitre.
* form\_label() affiche le label HTML du champ donné en argument. Le deuxième argument est le contenu du label.
* form\_widget() affiche le champ HTML lui-même (que ce soit<input>,<select>, etc.).
* form\_row() affiche le label, les erreurs et le champ en même temps, en respectant la vue définit dans le thème du formulaire que vous utilisez.
* form\_rest() affiche tous les champs manquants du formulaire (dans notre cas, juste le champ CSRF puisque nous avons déjà affiché à la main tous les autres champs).
* form\_end() affiche la balise de fermeture du formulaire HTML, soit</form>.

# Doctrine (Requêtes en BDD)

<https://symfony.com/doc/3.3/doctrine.html>

(Voir pdf)

C’est un ORM : Object Relational Mapping – Plus de requêtes en BDD, tous passe par des objets. Sorte de remplacement d’une BDD. Facilite le CRUD

Point important du fichier .env :

* la variable APP\_ENV a changé selon l’environnement ( dev,test,prod)
* la variable DATABASE\_URL

php bin/console doctrine :database :create -> créer la bdd ( en encodage latin : penser a aller modif sur phpmyadmin en utf8)

Shema = configuration des entités, l’ensemble des entités et leur propiètés.

## Relation entre Entity :

@ORM\ (1) (2)(=vers table) inversedBy/mappedBy (3) (->nom\_table)

1. : OneToMany ManyToOne

: ManyToMany ManyToMany

: OneToOne OneToOne

ManyToOne mappedBy