# Bibliographie

## Tuto 1 :

**Application SymBlog :**

<https://www.youtube.com/watch?v=UTusmVpwJXo>

<https://www.youtube.com/watch?v=_cgZheTv-FQ>

<https://www.youtube.com/watch?v=e5udJTjbYzw>

<https://www.youtube.com/watch?v=_GjHWa9hQic>

## Tuto 2 :

**Application Recette :**

<https://www.udemy.com/course/creer-4-applications-avec-symfony-v4-evolution-progressive/learn/lecture/16700800#overview>

## Tuto 3 :

## Application Automobile :

<https://www.udemy.com/course/creer-4-applications-avec-symfony-v4-evolution-progressive/learn/lecture/16700800#overview>

# Projet SYMFONY et Contrôleur

## VIDEO :

<https://www.youtube.com/watch?v=UTusmVpwJXo>

## Installation projet Symfony 4 :

(après installation du programme symfony)

symfony new --full recette

composer create-project symfony/skeleton blog

composer create-project "symfony/website-skeleton:^4.0" blog

## Installation du module server en mode de dev :

(deprecated) composer require server –dev

composer require --dev symfony/web-server-bundle

## Lancer l'application dans le serveur (mode dev uniquement) :

**(après installation du programme symfony)**

**symfony server:start**

(deprecated – only if pcntl php option isn’t availlable) php bin/console server:run

php bin/console server:start

## Installation du module server en mode de dev sur Symfony 5 :

**Modifier composer.json :**

**"extra"**: {  
 **"symfony"**: {  
 **"allow-contrib"**: **false**,  
 **"require"**: **"4.4.\*"** }  
}

**Installer ensuite le serveur :**

composer require --dev symfony/web-server-bundle

**Lancer le serveur (fonctionne en mode dev et non en mode prod) :**

php bin/console server:run

## Lancer l’application en mode production :

**Fichier .env :**

APP\_ENV=prod

**Commande :**

php -S localhost:8000 -t public

## Créer un contrôleur :

php bin/console make:controller

# DOCTRINE

## Fichier de configuration Symfony :

.env

## Variable dans .env pour renseigner les informations de la base de donnée :

DATABASE\_URL=mysql://root:@127.0.0.1:3306/blog

## Création de la base de données

(après paramétrage de la base de données dans .env)

php bin/console doctrine:database:create

## Créer une entité dans Symfony :

php bin/console make:entity User

## Création du fichier de migration :

php bin/console make:migration

## Lancement du ou des fichiers de migration pour mettre à jour la base de données :

php bin/console doctrine:migrations:migrate

**ATTENTION :**

**php bin/console doctrine:migrations:migrate --no-interaction**

## Installation du module fixtures en mode dev :

composer require orm-fixtures --dev

## Création d'une fixture :

php bin/console make:fixtures

### ATTENTION :

**Modifier l’import :**

use Doctrine\Common\Persistence\ObjectManager;

**par l’import :**

use Doctrine\Persistence\ObjectManager;

## Lancer les fixtures (après écriture du code dans la ou les classes fixtures de Doctrine) :

php bin/console doctrine:fixtures:load

**ATTENTION :**

**php bin/console doctrine:fixtures:load --no-interaction**

# Upload file

## Plugin Symfony de la communauté :

Chercher le plugin Vich :

<https://flex.symfony.com/>

## Installation :

composer require vich/uploader-bundle

## Configuration :

**Fichier config/bundles.php :**

Vich\UploaderBundle\VichUploaderBundle::class => ['all' => true],

**Fichier config/packages/vich\_uploader.yaml :**

vich\_uploader:

db\_driver: orm

mappings:

aliment\_image:

uri\_prefix: /images/aliments

upload\_destination: '%kernel.project\_dir%/public/images/aliments'

namer: Vich\UploaderBundle\Naming\SmartUniqueNamer

## Utilisation :

**Classe entité Aliment :**

/\*

\* @Vich\Uploadable\*

\*/

class Aliment

{

…

}

**Attribut pour upload image dans entité Aliment :**

/\*\*

\* @Vich\UploadableField(mapping="aliment\_image", fileNameProperty="image")

\*/

private $imageFile;

**Imports dans le fichier Entity Aliment :**

use Symfony\Component\HttpFoundation\File\**File**;

use Symfony\Component\HttpFoundation\File\**UploadedFile**;

use Vich\UploaderBundle\Mapping\Annotation as **Vich**;

# Formulaire

## VIDEO :

<https://www.youtube.com/watch?v=_cgZheTv-FQ>

## Créer un formulaire :

php bin/console make:form

## Utiliser thème bootstrap pour les formulaires :

**Documentation :**

<https://symfony.com/doc/current/form/bootstrap4.html>

**Modifier le fichier « config/packages/twig.yaml » :**

**twig**:  
 **default\_path**: **'%kernel.project\_dir%/templates'  
 form\_themes**: [bootstrap\_4\_layout.html.twig]

**Indiquer le nom du thème dans le fichier Twig si pas indiqué dans le fichier twig.yaml (avant le body) :**

{% form\_theme form 'bootstrap\_4\_layout.html.twig' %}

# Entités, relations et données aléatoires

## VIDEO :

<https://www.youtube.com/watch?v=e5udJTjbYzw>

## Rappel : Créer une entité dans Symfony :

(vérifier si cette commande est vraiment utile) composer require orm

php bin/console make:entity

## Renseigner une relation dans l’entité en cours de création :

**Exemple :**

Entity : « category »

Add property : « articles » car relation OneToMany entre « category » et « article »

Field type : « relation » car il faut créer le lien entre les 2 entités

Related to : « Article » car table en relation

Relation type : « OneToMany »

New field name on inverse table : « category » is default value (type enter key here)

…

**Lancer ensuite :**

php bin/console make:migration (ne pas oublier dans cet exemple de supprimer les enregistrements articles car le champ « category » ne peut pas être null : voir exemple précédent).

php bin/console doctrine:migrations:migrate

## Installer la librairie « faker »

composer require fzaninotto/faker –dev

## Création fixture (rappel) :

**Ecrire le code puis lancer les fixtures :**

php bin/console doctrine:fixtures:load

# Sécurité

## VIDEO :

<https://www.youtube.com/watch?v=_GjHWa9hQic>

## Fichier de configuration :

config/packages/security.xaml

Il faudra configurer les sections :

* encoders
* providers
* firewalls

## Création de l’entité « User »

php bin/console make:entity User

php bin/console make:migration

php bin/console doctrine:migrations:migrate

## Création du formulaire d’inscription

php bin/console make:form RegistrationType

## Création du contrôleur

php bin/console make:controller

## Définir l’encodage

**Dans le fichier de configuration « config/packages/security.yaml » :**

encoders:

App\Entity\User:

algorithm: bcrypt

**L’entity User doit implementer l’interface « UserInterface » :**

**use** Symfony\Component\Security\Core\User\UserInterface;

**Important :**

Définir les méthodes manquantes et demandées par l’interface « UserInterface ».

## Paramétrer le provider

**Dans le fichier de configuration « config/packages/security.yaml » :**

providers:

in\_database:

entity:

class: App\Entity\User

property: email

## Paramétrer le firewall

**Dans le fichier de configuration « config/packages/security.yaml » :**

**LOGIN**

firewalls:

main:

anonymous: lazy

provider: in\_database

form\_login:

login\_path: security\_login

check\_path: security\_login

**LOGOUT**

firewalls:

main:

anonymous: lazy

provider: in\_database

logout:

path: security\_logout

target: home

## Contrôle des accès aux pages

**Paramétrer les contrôles d’accès en fonction du rôle de l’utilisateur dans le fichier de configuration « config/packages/security.yaml » :**

Dans cet exemple, les url commençant par /admin ne seront accessible que pour les utilisateurs ayant le rôle ROLE\_ADMIN

access\_control:

- { path: ^/admin, roles: ROLE\_ADMIN }

## Créer un formulaire de commentaire pour utilisateurs identifiés

php bin/console make:form

# Gestion des erreurs 404

## Lancer l’application en mode production (rappel) :

**Fichier .env :**

APP\_ENV=prod

**Commande :**

php -S localhost:8000 -t public

## Installer bundle « twig » (si nécessaire) :

composer require twig

## Créer page 404 :

**Path du fichier :**

templates/bundles/TwigBundle/Exception/error404.html.twig

# Traduction messages d’erreur :

## Paramétrer la langue d’affichage des messages d’erreur :

**Fichier config/packages/translation.yaml :**

framework:

default\_locale: fr

translator:

default\_path: '%kernel.project\_dir%/translations'

fallbacks:

- fr

## Récupérer message d’erreur dans le contrôleur :

/\*\*

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

\*/

public function login(AuthenticationUtils $util)

{

return $this->render("security/login.html.twig", [

"lastUserName" => $util->getLastUsername(),

**"error" => $util->getLastAuthenticationError()**

]);

}

## Afficher message d’erreur dans vue :

**Sans traduction :**

{% if error %}

<div class="alert alert-danger">

{{ error.messageKey }}

</div>

{% endif %}

**Avec traduction :**

{% if error %}

<div class="alert alert-danger">

{{ error.messageKey | trans(error.messagedata, "security") }}

</div>

{% endif %}

# Pagination

## Installation plugin

**URL du plugin :**

<https://github.com/KnpLabs/KnpPaginatorBundle>

**Commande :**

composer require knplabs/knp-paginator-bundle

## Configuration

**Créer fichier de configuration :**

config/packages/knp\_ paginator.yaml

**Paramètres de base (voir documentation du plugin) :**

**knp\_paginator**:  
 **page\_range**: 5 *# number of links showed in the pagination menu (e.g: you have 10 pages, a page\_range of 3, on the 5th page you'll see links to page 4, 5, 6)* **default\_options**:  
 **page\_name**: page *# page query parameter name* **sort\_field\_name**: sort *# sort field query parameter name* **sort\_direction\_name**: direction *# sort direction query parameter name* **distinct**: true *# ensure distinct results, useful when ORM queries are using GROUP BY statements* **filter\_field\_name**: filterField *# filter field query parameter name* **filter\_value\_name**: filterValue *# filter value query parameter name* **template**:  
 **pagination**: **'@KnpPaginator/Pagination/ sliding.html.twig'***# sliding pagination controls template* **sortable**: **'@KnpPaginator/Pagination/sortable\_link.html.twig'** *# sort link template* **filtration**: **'@KnpPaginator/Pagination/filtration.html.twig'** *# filters template*

## Utilisation

**Dans le repository écrire la méthode qui renvoie un object de type Query au lieu de renvoyer un résultat :**

**public function** findAllWithPagination() : Query {  
 **return** $this->createQueryBuilder(**'v'**)  
 ->getQuery();  
}

**Méthode dans le contrôleur :**

*/\*\*  
 \** ***@Route****("/", name="view\_all")  
 \*/***public function** viewAll(VoitureRepository $repository, PaginatorInterface $paginator, Request $request)  
{  
 $voitures = $paginator->paginate(  
 $repository->findAllWithPagination(), */\* query NOT result \*/* $request->**query**->getInt(**'page'**, 1), */\*page number\*/* 8 */\*limit per page\*/*);  
  
 **return** $this->render(**'voiture/view\_all.html.twig'**,[  
 **"voitures"** => $voitures  
 ]);  
}

**Code dans le fichier twig pour afficher la pagination :**

*{# display navigation #}*<**div class="navigation"**>  
 {{ knp\_pagination\_render(voitures) }}  
</**div**>

**Modifier le design de la pagination :**

**Remplacer la ligne du fichier de configuration :**

**pagination**: **'@KnpPaginator/Pagination/sliding.html.twig'**

**Par la ligne suivante (voir la documentation du plugin) :**

**pagination**: **'@KnpPaginator/Pagination/twitter\_bootstrap\_v4\_pagination.html.twig'**

## Traduction du bundle pour affichage de la pagination :

**Créer le fichier :** « translations/KnpPaginatorBundle.fr.yaml »

**Renseigner le fichier :**

**label\_next**: Suivant  
**label\_previous**: Précédent

**Dans le fichier « config/packages/translation.yaml » :**

**framework**:  
 **default\_locale**: fr  
 **translator**:  
 **default\_path**: **'%kernel.project\_dir%/translations'  
 fallbacks**:  
 - fr

# Filtres

## Créer l’Entity pour filtrer la recherche

**ATTENTION :**

Cette entité n’est pas sauvegardée en base de données : sa création se fait manuellement et non en ligne de commande.

## Créer le formulaire pour la recherche

**ATTENTION :**

Lors de la création du formulaire avec la commande « php bin/console make:form » pour la question concernant l’entité il faut indiquer le chemin complet car ici l’entité n’est pas renseignée en base de données.

Exemple :

php bin/console make:form RechercheVoitureType

\App\Entity\RechercheVoiture

## Déploiement de l’application

## Installation du serveur Apache

composer require symfony/apache-pack

## Passage du mode de développement en mode production

**Dans le fichier .env remplacer :**

**APP\_ENV**=**dev**

**Par :**

**APP\_ENV**=**prod**

**Et :**

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

**Par :**

**DATABASE\_URL**=**<Paramètres de la base de données de production>**

## Transfert vers l’hébergeur

1. Vider le cache de l’application
2. Transfert
3. Création de la base de données
4. Créer un sous-domaine qui pointe vers le répertoire « public » de l’application
5. Enjoy… 😉