**Définition du projet**

Le site doit présenter différents types et modèles de guitares.

Il sera possible de faire des recherches par type et par marque.

Chaque guitare sera présentée sur une page et aura une description et des attributs techniques

**Conception du MCD / MLD**

(copies d’écran)

**Lancement du projet**

Le projet est lancé dans le dossier www

\*Vérifier que composer est bien installé https://getcomposer.org/download/

Téléchargement de Symfony « Guitares »

Lancement dans l’invit de commande / Commandes successives :

*Créer le dossier d’install Symfony*

php -r "file\_put\_contents('symfony', file\_get\_contents('https://symfony.com/installer'));"

*Création du projet*

php symfony new Guitars

*Installer les dépendances*

composer install

*Générer le bundle*

php bin/console generate:bundle

!!! Créer la BDD dans phpMyAdmin !!!

*Générer les entités avec Doctrine*

php bin/console doctrine:generate:entity

Création des entités dans l’invit de commande

Entités créées

**guitar**

id

serie

description

picture

date\_article

video\_link

seo\_title

seo\_desc

micros

body\_wood

fretboard\_wood

neck\_wood

**author**

Id

nom

**brand**

Id

name

**comment**

Id

content

date\_comment

**newsletter**

Id

mail\_user

**subtype1**

Id

Name

**subtype2**

Id

name

**tag**

Id

name

**type**

Id

name

**user**

Id

user\_pseudo

user\_email

* Mettre à jour le schéma et voir si la création de la base a bien fonctionné
  + - php bin/console doctrine :schema :validate
* Un message d’erreur s’affiche « FAIL – The database schema is not in sync with the current … »
  + - php bin/console doctrine :schema :update --force
* **La base est créée dans PhpMyAdmin !**

Une fois les entités créées, il convient de faire les relations (liens) entre chacune de façon à récupérer les clés étrangères.

**Créer les relations bidirectionnelles suivant le modèle ci-dessous :**

**ManyToOne**

*/\*\**

*\* Many Products have One Category.*

*\* @ORM\ManyToOne(targetEntity="Category", inversedBy="products")*

*\* @ORM\JoinColumn(name="category\_id", referencedColumnName="id")*

*\*/*

private $category;

**OneToMany**

*/\*\**

*\* One Category have Many Products.*

*\* @ORM\OneToMany(targetEntity="Product", mappedBy="category")*

*\*/*

private $products;

public function \_\_construct()

{

$this->products = new ArrayCollection();

}

Et rajouter en haut de fichier le use car un tableau sera généré et il y aura des données à stocker et à récupérer

Pour cela on utilise ArrayCollection qui est un composant de Doctrine

use Doctrine\Common\Collections\ArrayCollection;

**ManyToMany ( *!!! attention relation unidirectionnelle !!! )***

*/\*\**

*\* Many Users have Many Groups.*

*\* @ManyToMany(targetEntity="Group", inversedBy="users")*

*\* @JoinTable(name="users\_groups")*

*\*/*

private $groups;

public function \_\_construct() {

$this->groups = new ArrayCollection();

}

Et rajouter en haut de fichier le use car un tableau sera généré et il y aura des données à stocker et à récupérer

Pour cela on utilise ArrayCollection qui est un composant de Doctrine

use Doctrine\Common\Collections\ArrayCollection;

Mettre à jour le schéma doctrine

php bin/console doctrine :schema :update --force

**ICI** on fait **le getter** de l’attribut **tags** dans l’entité **guitar**

Le getter permet de récupérer l’objet tag.

On y ajoute des fonctions REMOVE et ADD car on doit le faire pour les relations X/ToMany

Pour les autres relations (X/ToOne), il faut faire un getter classique

/\*\*  
\* Many Guitars have Many Tags.  
 \* @ORM\ManyToMany(targetEntity="Tag")  
\* @ORM\JoinTable(name="guitars\_tags",  
 \* joinColumns={@JoinColumn(name="guitar\_id", referencedColumnName="id")},  
 \* inverseJoinColumns={@JoinColumn(name=tag\_id", referencedColumnName="id")})  
 \*/  
   
 private $tags;  
  
 public function \_\_construct() {  
 $this->tags = new ArrayCollection();  
 $this->comments = new ArrayCollection();  
 }  
  
  
/\*\*  
\* Add tag  
\*  
 \* @param Tag $tag  
\*  
 \* @return Tag  
 \*/  
 public function addTag(Tag *$tag*)  
{  
 if (!$this->tags->contains(*$tag*)) {  
 $this->tags->add(*$tag*);  
 *$tag*->setGuitar($this);  
 }  
  
 return $this;  
}  
  
 /\*\*  
 \* Remove tag  
 \*  
 \* @param Tag $tag  
 \*  
 \* @return Tag  
 \*/  
 public function removeTag(Tag *$tag*)  
{  
 if ($this->tags->contains(*$tag*)) {  
 $this->tags->removeElement(*$tag*);  
 *$tag*->setGuitar(null);  
 }  
  
 return $this;  
}  
  
 /\*\*  
 \* Get tags  
 \*  
 \* @return ArrayCollection  
 \*/  
 public function getTag()  
{  
 return $this->tags;

**CREATION DES ROUTES**

La route pour la page d’accueil sera créée dans le HomeController (initialement DefaultController).

Le controller

**<?php  
  
namespace** SiteBundle\Controller;  
  
**use** Symfony\Bundle\FrameworkBundle\Controller\Controller;  
**use** Sensio\Bundle\FrameworkExtraBundle\Configuration\Route;  
  
**class** HomeController **extends** Controller  
{  
 /\*\*  
 \* Matches / exactly  
 \*  
 \* @Route("/", name="home\_home")  
 \*/  
 **public function** homeAction()  
 {  
 **return** $this**->**render('SiteBundle:Default:index.html.twig'); => renvoie l’affichage  
 }  
}

Ensuite on va tester l’affichage => !!!!! PENSER A PASSER $kernel en true

**<?php  
  
use** Symfony\Component\HttpFoundation\Request;  
  
/\*\* @var \Composer\Autoload\ClassLoader $loader \*/  
$loader **= require** \_\_DIR\_\_**.**'/../app/autoload.php';  
**include\_once** \_\_DIR\_\_**.**'/../var/bootstrap.php.cache';  
  
$kernel **= new** AppKernel('prod', **true**);

On crée le controller BrandController, puis une route avec l’action associée qui renvoie à la view que l’on crée ci-dessous.

On crée un fichier brand.html.twig dans le dossier views\Public pour l’affichage de la partie brand

On crée une boucle FOR pour récupérer les guitares associées à une marque pour les afficher quand on va pointer sur la page de la marque.